UNIVERSIDADE DE CAXIAS DO SUL

JÓRDAN RUI DA ROSA

CRIAÇÃO E IMPLANTAÇÃO DE UM DATA WAREHOUSE PARA REFORMATAR O MECANISMO DE PESQUISA DO PORTAL INTERGENICDB 2.0

CAXIAS DO SUL
2016

JÓRDAN RUI DA ROSA

CRIAÇÃO E IMPLANTAÇÃO DE UM DATA WAREHOUSE PARA REFORMATAR O MECANISMO DE PESQUISA DO PORTAL INTERGENIC 2.0

Trabalho de Conclusão de Curso do Título de Bacharel do Curso de Ciência da Computação pela Universidade de Caxias do Sul. Área de concentração: evolução de software.

Orientador Prof. Me. Daniel Luis Notari

CAXIAS DO SUL
2016

À minha esposa, Caroline, que me cuidou, me acalmou e me amou. Esse trabalho somente obteve sucesso graças à sua dedicação e carinho.

AGRADECIMENTOS

Agradeço, primeiramente à minha esposa, Caroline Domanski Dall'Acua. Sua dedicação, carinho e amor foram fundamentais nessa etapa da minha vida. Suas palavras, seus abraços e beijos me motivaram, me dando energia para chegar ao fim dessa estrada.

Meus sinceros agradecimentos ao meu orientador, Daniel Notari, que me acolheu como seu orientado. Graças a ele, e à Prof. Sheila de Ávila e Silva, tive a oportunidade de trabalhar nesse projeto. Obrigado pelos conselhos, pelo tempo, pela paciência e pela atenção que dedicaram a mim nesses dias.

Sou muito grato aos meus pais, Didimo Palhano Da Rosa e Maria Teresa Rui, que, juntamente com meus amigos e demais familiares, compreenderam minha ausência em muitos dias atípicos. Obrigado pela paciência, pelo carinho, pelas palavras de apoio, pela torcida e todas as energias positivas que recebi de vocês.

Não posso deixar de agradecer também a todos os professores, queridos mestres com os quais, durante esses inúmeros semestres, compartilhei noites e mais noites de estudos. Obrigado por transmitirem seus conhecimentos, por dedicarem seu tempo e amor à profissão. Se estou escrevendo essas palavras, é porque vocês me mostraram como chegar até aqui.

Também gostaria de agradecer a colega, e nova amiga, Camila Rachel Tonin de Andrade. Seu apoio, dedicação e paciência durante esta trajetória foram de grande ajuda. Estou imensamente grato por nossos caminhos terem se cruzado e por termos trabalhados juntos nesse projeto

O errado é errado, mesmo que todo mundo esteja fazendo. O certo é certo, mesmo que ninguém esteja fazendo.

Autor desconhecido

Insanidade é continuar fazendo sempre a mesma coisa e esperar resultados diferentes

Albert Einstein

RESUMO

A biologia molecular é uma área da biologia que estuda as células em nível molecular, tendo um dogma central focado no estudo da síntese de proteínas. A região intergênica e seus promotores são fatores reguladores no processo, contendo as informações essenciais. Dessa forma, a bioinformática utiliza meios computacionais para solucionar problemas da biologia molecular. Um exemplo disso é o banco de dados PromotoresDB – que armazena sequências de nucleotídeos de regiões promotoras de organismos procariontes – e o portal IntergenicDB – que possui ferramentas de acesso e gestão dos dados. Ambos foram criados para atender às necessidades do grupo de Bioinformática da Universidade de Caxias do Sul. Porém, o portal e a base de dados têm apresentado instabilidades na sua ferramenta de consulta. Outro fator, como a interface, também afeta a ferramenta atualmente utilizada. Diante de tais problemáticas, o objeto desse trabalho é remodelar a ferramenta, fazendo alterações em modelos, banco de dados, código fonte e visual, buscando o melhor desempenho e recursos na ferramenta para realizar consultas à base de dados. Durante a execução da proposta de solução, a equipe realizou uma evolução do sistema, migrando do MySQL para o PostgreSQL e do C# para o PHP, dando origem ao projeto IntergenicDB 2.0, solucionando os problemas detectados. A proposta deste trabalho foi mantida e executada na nova versão como parte da solução. Os tempos de execução através de um *Data Warehouse* são efetivamente superiores do que a utilização de *view*. Em bases pequenas, com poucas linhas e tamanho, essa diferença é imperceptível. O PromotoresDB não é uma base com crescimento diário, mas a curva terá um comportamento exponencial quando realizada a inserção de dados. A reformatação da tela de pesquisa possibilita ao usuário refinar as consultas montando graficamente os filtros, resultando numa consulta em SQL ao banco de dados. Requer um conhecimento prévio de operadores lógicos e, se possível, noções de SQL irão permitir uma melhor experiência ao utilizador. A tabela na tela de resultado possui uma formatação minimalista para melhor visualização dos dados, com fundo branco, demais composições em tons de cinza e destaques realizados com o mouse nos tons do tema do portal. O resultado da pesquisa é paginado, conforme parametrizado na tela de pesquisa, permitindo que a navegação seja rápida e eficaz. Em necessidades especificas como formatações e

outros tipos de filtros, por exemplo, tem-se a possibilidade de fazer o download de todas as linhas em um arquivo de texto. O formato CSV permite a importação dos dados para outros sistemas e programas como, por exemplo, planilha de cálculo e assim realizar tais manipulações.
Palavras-Chave: IntergenicDB. Biologia Molecular. Bioinformática. Regiões Intergênicas. PromotoresDB. Promotores. Banco de Dados de Biologia Molecular. *Data Warehouse*.

ABSTRACT

Molecular biology is an area which studies cells in molecular level, with a central dogma aiming protein synthesis. Intergenic region and their promoters are regulators on this process containing essential information. Therefore bioinformatics uses computational resources to solve problems in molecular biology. For instance, database from PromotoresDB – stores nucleotides sequences from prokaryotes organisms' promoters regions – and the IntergenicDB portal – contains access tools and data management. Both were created to support demands from Bioinformatics group from Caxias do Sul University. However, the portal and database has presented instability on its search tools. Another factor, as interface, also affects currently used tool. Against these problematics the goal of this work is to remodel the tool, making alterations in models, database, source code and layout, aiming the best performance and resources in the tool to make search at the database. When executing the solution proposal, the team made an evolution on the system, migrating from MySQL to PostgreSQL as well as from C# to PHP, originating the project Intergenic 2.0 and solving detected problems. The proposal from this work was kept and executed on the new version as a part of the solution. The execution times through Data Warehouse are effectively superior to view uses. In small bases, such few lines and size, this difference is imperceptible. The PromotoresDB is not a daily increasing database, but the curve will present an exponential behaviour when data insertion is made. Reformatting the screen search allows the user to refine the research assembling the filters graphically, resulting in a SQL search on database. It requires previous knowledge from logical operator and, whenever possible, SQL basic ideas which will allow the best experience for the user. The chart on result screen presents minimalist formation to a better data view, in a white background with other components in grey tones and the highlights made from the mouse on portal theme. The result of the research is paged, as patterned on research screen, allowing efficient and fast use. Specific needs as formatting and other types of filters, for instance, there is the possibility to download the lines in a text file. The CSV shape allows data import to other systems and programs as, for example, a spreadsheet and also manipulate it.

Key-words: IntergenicDB. Molecular biology. Bioinformatics. Intergenic regions. PromotoresDB. Promoters. Database from Molecular Biology. Data Warehouse.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADN	Ácido desoxirribonucleico
ARN	Ácido ribonucleico
ASP	*Active Server Pages*
BD	Banco de Dados
BDBM	Banco de Dados de Biologia Molecular
C#	*C Scharp*
CI	*CodeIgniter*
CR	*Carriage Return*
CSS	*Cascading Style Sheets*
DDBJ	*DNA DataBank Japan*
DNA	*Deoxyribonucleic acid*
DW	*Data Warehouse*
EF	*Entity Framework*
EMBL	*European Molecular Biology Laboratory*
EMBL-EBI	*European Bioinformatics Institute*
ENA	*European Nucleotide Archive*
ETL	*Extract, transform and load*
EUA	Estados Unidos da América
FK	*Foreign Key*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
I1	*IntergenicDB1.0*
I2	*IntergenicDB2.0*
IDE	*Integrated Development Environment*
IIS	*Internet Information Services*
IIS	*Internet Information Services*
INSDC	*International Nucleotide Sequence Database Collaboration*
LF	*Line Feed*
MMS	*Mass Submission System*
mRNA	RNA mensageiro

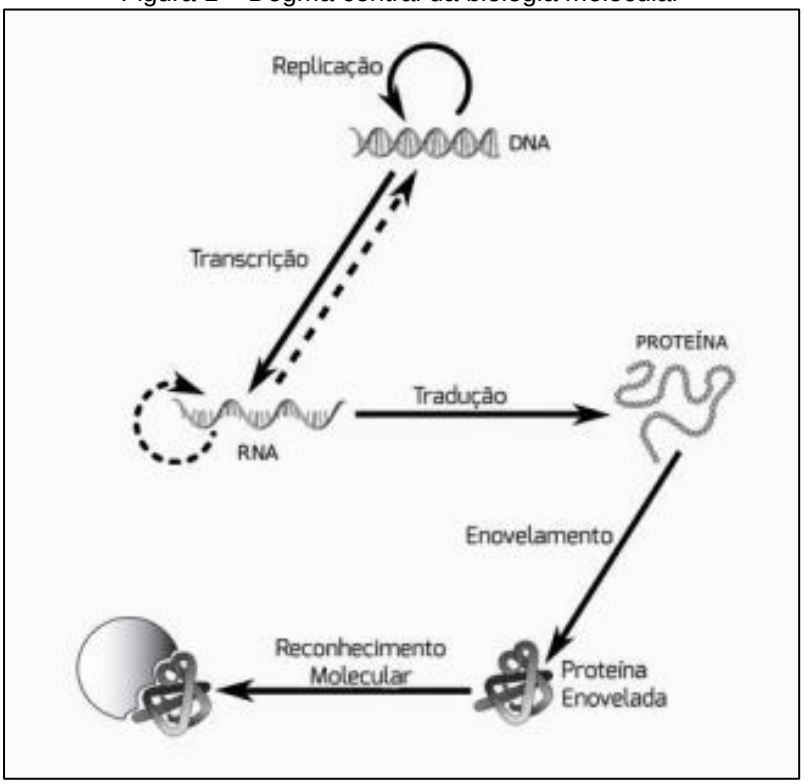
MVC	*Model-view-controller*
NCBI	*National Center for Biotechnology Information*
NIH	*National Institutes of Health*
NLM	*National Library of Medicine*
PHP	*Hypertext Preprocessor*
PM	*Planner Method*
POO	*Programação Orienta a Objetos*
RNA	*Ribonucleic acid*
rRNA	RNA ribossômico
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
tRNA	RNA transportador
UCS	Universidade de Caxias do Sul
VM	*Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO

A biologia molecular é uma área da biologia que estuda os fenômenos das interações química e física das células em nível molecular. Um aspecto importante para a biologia molecular é o dogma central da biologia molecular, ilustrado na Figura 1, que trata dos três tipos de moléculas mais analisados pela bioinformática: o DNA, o RNA e as proteínas (COX; DOUDNA; O'DONNEL, 2012; VERLI, 2014).

Figura 1 – Dogma central da biologia molecular



Fonte: Verli (2014, p. 15).

O ponto fundamental do dogma é a produção de proteínas, que são cadeias de aminoácidos fundamentais para as células dos seres vivos. Suas funções incluem suporte estrutural, proteção, transporte, regulação, movimento, entre outros. A chave para a síntese de proteínas, conforme apresentado na Figura 1, são os ácidos nucleicos, também conhecidos por DNA e RNA, além dos processos de transcrição e tradução (SADAVA et al., 2009; ALBERTS et al., 2010; GOWDAK et al., 2013).

A região intergênica, a qual contém seus promotores, atua como agentes reguladores no processo de transcrição. São fundamentais pois contém as

informações que regulam o processo de transcrição, passo final para sintetizar as proteínas (ALBERTS et al., 2010; ZAHA; FERREIRA; PASSAGLIA, 2014).

A complexidade e a quantidade de informações envolvidas no dogma central da biologia geraram a necessidade da criação de recursos computacionais capazes de armazenar, organizar, processar e filtrar tamanha cadeia de informações. Para sanar essas questões, nasceu a bioinformática (VERLI, 2014; ZAHA; FERREIRA; PASSAGLIA, 2014).

A bioinformática é um novo ramo da ciência que utiliza técnicas computacionais para suprir essas necessidades da área da biologia molecular, criando equipamentos e soluções desde as estruturas de hardware até desenvolvimentos e melhorias em software (VERLI, 2014; ZAHA; FERREIRA; PASSAGLIA, 2014).

As melhorias de software são necessárias quando a ferramenta atual não atende como, por exemplo, os requisitos atuais, novas demandas, mudanças de hardware e também para acompanhar as mudanças do sistema operacional. É de competência da Engenharia de Software esta área do conhecimento chamada de Evolução de Software, a qual busca teorizar e fornecer mecanismos para as constantes necessidades que um sistema tem de se adaptar durante sua vida útil (SOMMERVILLE, 2011).

1.1 PROBLEMA DE PESQUISA

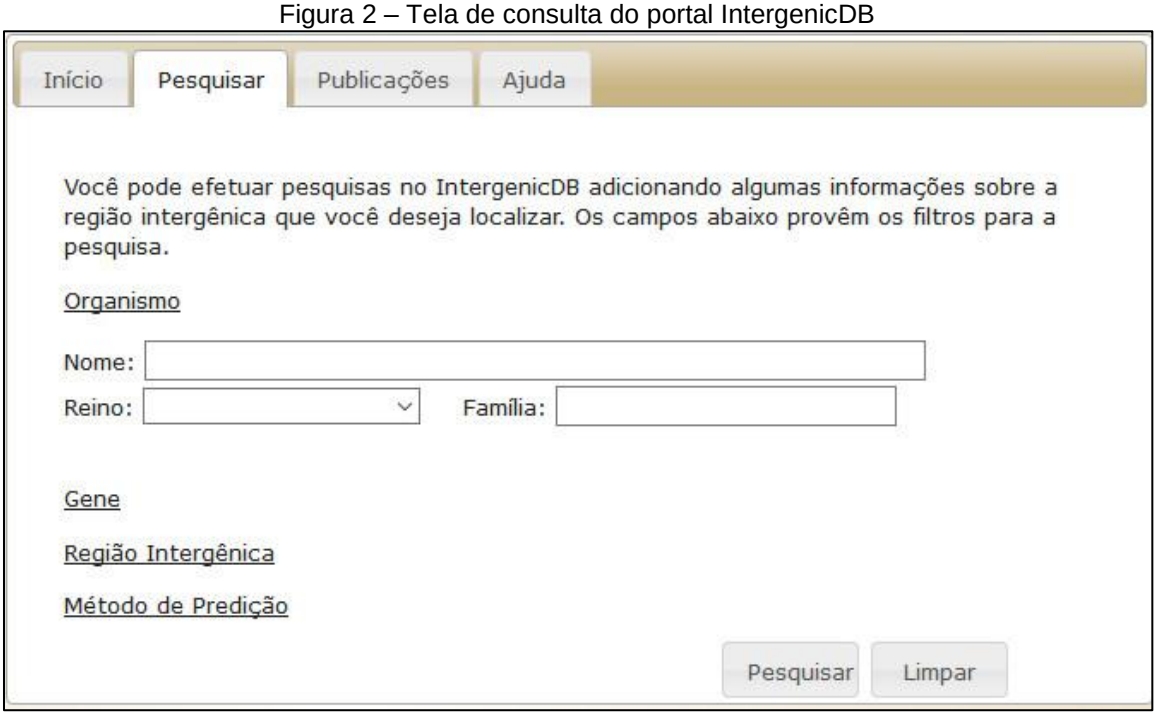
O portal IntergenicDB é um repositório de acesso público, no qual, pesquisadores podem ter acesso às informações do banco de dados PromotoresDB. Por não haver meios práticos de extrair as regiões intergênicas dos bancos de dados públicos da atualidade, foi desenvolvido o portal. Seu objetivo é integrar os dados de regiões intergênicas com as informações dos seus genes (DALZOCHIO, 2014; NOTARI et al., 2014)

O banco de dados é povoado com informações inseridas por usuários cadastrados através, e principalmente, pela importação de dados de outras bases públicas. Todas essas informações novas precisam ser aprovadas por usuários com nível de administrador. A centralização dessas informações facilitará o estudo de

sequências intergênicas com possível função biológica de DNA de organismos procariontes (DANVANZO,2010).

A ferramenta de pesquisa do portal, mostrada na Figura 2, viabiliza a consulta das informações contidas na base de dados, através do preenchimento prévio dos campos. Não é necessário preencher todos os campos, mas para realizar a pesquisa pelo menos um dos campos deve estar preenchido. O preenchimento de mais campos permitirá que o motor de busca cruze as informações, trazendo dados mais refinados.

Figura 2 – Tela de consulta do portal IntergenicDB



Fonte: IntergenicDB[1] (2015).

Atualmente, ao realizar uma pesquisa, o usuário irá se deparar com alguns problemas com a ferramenta:

a) lentidão ou *timeout* para exibir um resultado de pesquisa;
b) o resultado exibido não corresponde com os parâmetros inseridos;
c) a tela com o resultado da pesquisa possui uma interface não amigável, com poucos recursos;
d) cada registro decorrente da pesquisa é separado em páginas diferentes;
e) diversidade nos formatos de arquivos para *download*;
f) *framework* de consulta complexo, dificultando a identificação de falhas e possíveis correções.

[1] Disponível em: <http://intergenicdb.bioinfoucs.com/Default/Search>. Acessado em 22/11/2015.

Tais problemas como, rotulam o portal com aspectos negativos e dificultam sua divulgação e expansão dentro da área da biologia molecular. Uma vez em que o utilizador requer os dados e não consegue obtê-los, seja por um problema da lentidão (muitas vezes compreensível) ou pela inconsistência dos dados resultantes da pesquisa. Esse aspecto é o mais crítico de todos os listados acima.

1.2 QUESTÃO DE PESQUISA

Baseado no problema de pesquisa descrito anteriormente, foi criada a seguinte questão de pesquisa: quais melhorias são necessárias para que a ferramenta de busca funcione correta e adequadamente?

1.3 OBJETIVO

1.3.1 Objetivo geral

O objetivo deste trabalho é desenvolver e aplicar, para o banco de dados PromotoresDB, um *Data Warehouse* e, para o portal IntergenicDB 2.0, elaborar um mecanismo de pesquisa compatível com a melhoria proposta, resolvendo as instabilidades do portal IntergenicDB 1.0 atual. Além disso, serão feitos os ajustes necessários na aplicação existente e no banco de dados PromotoresDB.

1.3.2 Objetivos específicos

a) Definir os novos requisitos para a solução;
b) Implementar um novo mecanismo de busca;
c) Otimização do tempo de uma consulta ao bando de dados

d) Redesenhar as telas tornando as compatíveis com os novos mecanismos;
e) Padronizar e unificar os formatos de arquivo para *download*;
f) Verificar a consistência dos requisitos através de testes em ambiente controlado;
g) Validar o trabalho através dos testes.

1.4 ORGANIZAÇÃO DO TRABALHO

O capítulo 2 apresenta uma introdução à biologia molecular. Nas primeiras sessões são conceituados o dogma central da biologia molecular e demais elementos que pertençam ao tema. Tais dados são relevantes para a compressão da região intergênica e dos promotores.

As sessões intermediárias tratam da Bioinformática, descrevendo a área e os objetos. São conceituadas duas ferramentas – banco de dados e portais de biologia molecular – que são tema desse trabalho.

As últimas sessões descrevem três portais de biologia molecular e suas ferramentas de pesquisa.

O capítulo 3 apresenta o portal IntergenicDB e a base de dados PromotoresDB. São relatados os modelos, estruturas e tecnologias utilizadas no desenvolvimento de ambos os produtos da bioinformática.

O capítulo 4 explica a proposta de solução. Nela estão descritos os problemas encontrados, a solução com os requisitos e os testes que serão executados para validar a proposta.

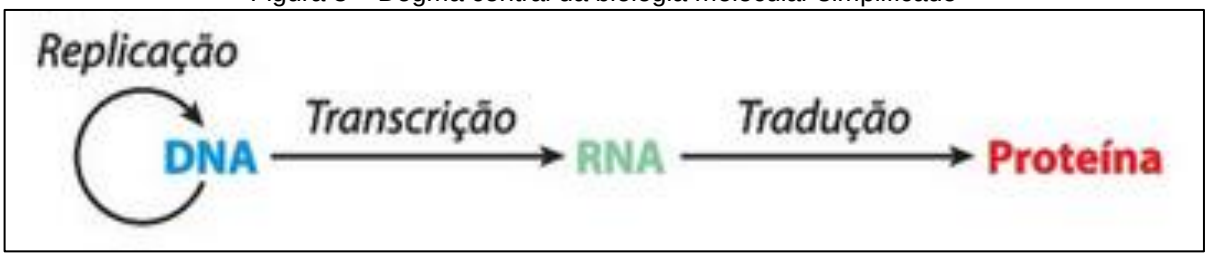
O capítulo 5 detalha o desenvolvimento, os testes e resultados propostos no capitulo 4. As subseções estão estruturadas semelhantes ao capítulo anterior, para facilitar a compreensão.

2 BIOLOGIA MOLECULAR

A Biologia Molecular é uma área da biologia que estuda as macromoléculas celulares essenciais e suas relações químicas e físicas, direcionando seu foco ao que os especialistas da área chamam de dogma central da biologia molecular (Figura 3).

A importância desse tópico é que ele trata dos três tipos de macromoléculas celulares mais usados pela bioinformática – o DNA, o RNA e as proteínas – e dos processos fundamentais para transportar de forma confiável as informações genéticas de uma geração para outra (COX; DOUDNA; O'DONNEL, 2012; VERLI, 2014).

Figura 3 – Dogma central da biologia molecular simplificado



Fonte: Sanders; Bowman (2014, p. 10).

Nesse capítulo, serão apresentados os conceitos básicos e necessários para compreender o dogma central da biologia molecular e sua relação com o objetivo proposto por esse trabalho.

2.1 CÉLULAS

A célula é a menor unidade viva da complexa estrutura que forma um organismo vivo, retendo suas características morfológicas e fisiológicas. Basicamente, todas as células possuem membrana plasmática e citoplasma. A membrana serve para delimitar a célula, mas não isolar, funcionando como uma barreira seletiva de substâncias, permitindo ou impedindo a entrada e saída dessas substâncias. O citoplasma, parte estrutural da célula, é composto basicamente por substâncias como água, íons e algumas moléculas (SADAVA et al., 2009; ALBERTS et al., 2010; GOWDAK; MATTOS; PEZZI, 2013).

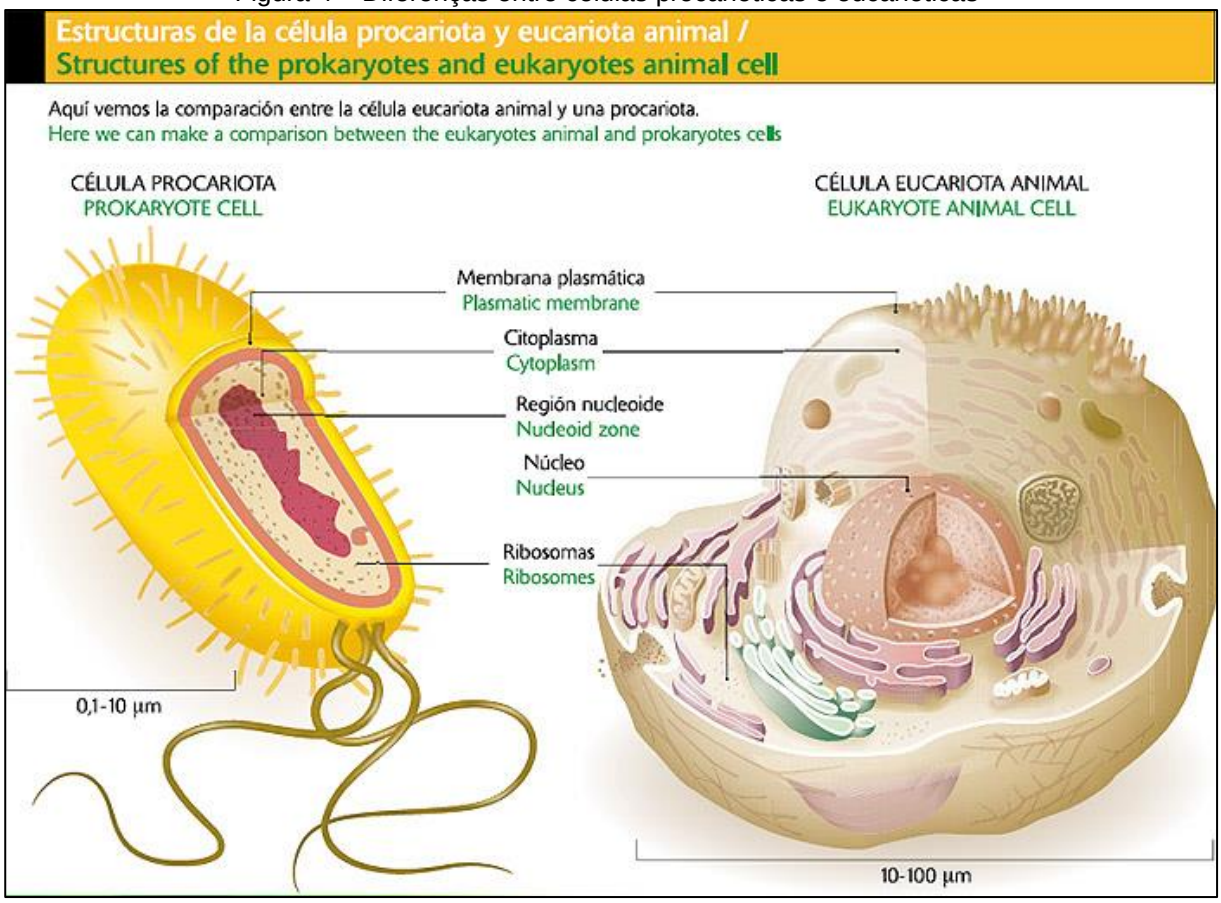
Conforme Sadava et al. (2009), a teoria celular fala que:

a) todos os seres vivos são formados por uma ou mais células;
b) as células nascem de outras células;
c) todas as células são semelhantes em composição química;
d) grande parte das reações químicas fundamentais à vida ocorrem dentro das células;
e) conjuntos de informações genéticas da vida estão dentro das células.

Embora haja alguns processos bioquímicos, moléculas e estruturas sendo compartilhadas por todos os tipos de células, elas são divididas em dois grupos principais: procariontes e eucariontes (SADAVA et al., 2009).

Segundo Sadava et al. (2009, p. 72), “Tanto células procarióticas como eucarióticas estão delimitadas por uma membrana plasmática, no entanto, as células procarióticas não possuem compartimentos internos delimitados por membrana”, ou seja, os procariontes não possuem invólucro nuclear. Essas diferenças podem ser vistas na Figura 4.

Figura 4 – Diferenças entre células procarióticas e eucarióticas



Fonte: Picolotto (2012 apud C.B.C, 2012, p. 13).

Outras características que diferenciam ambos, é que a maioria dos procariontes são unicelulares, não possuem citoesqueleto e poucas ou nenhuma organela. Entretanto, sua parede celular é mais complexa contendo mais moléculas do que nos eucariotos. A ausência do invólucro, nos procariotos, faz com que o genoma fique em contato direto com o citoplasma, o ribossomo, algumas partículas e moléculas (ALBERTS et al., 2010).

2.2 A GENÉTICA, OS GENES E OS GENOMAS

A genética é a área de estudo da hereditariedade, processo em que um genitor transfere suas características para a prole, fazendo com que o novo ser se assemelhe aos seus ancestrais. Os seres humanos são um exemplo disso. Essa herança genética é controlada por numerosos fatores: os genes (BROWN, 2009).

Os genes são pequenas partículas físicas presentes em todos os organismos vivos que contêm as informações de hereditariedade. Elas são segmentos de moléculas de DNA que trabalham no controle da transcrição genética e possuem informações para sintetizar moléculas de RNA e proteínas (BROWN, 2009; ALBERTS et al., 2010; SANDERS; BOWMAN, 2014).

O genoma é um grupo de genes com todas as informações genéticas das moléculas de DNA de um organismo. Esse conjunto é divido em dois grupos: as regiões gênicas e as regiões intergênicas (BROWN, 2009; ALBERTS et al., 2010; SANDERS; BOWMAN, 2014).

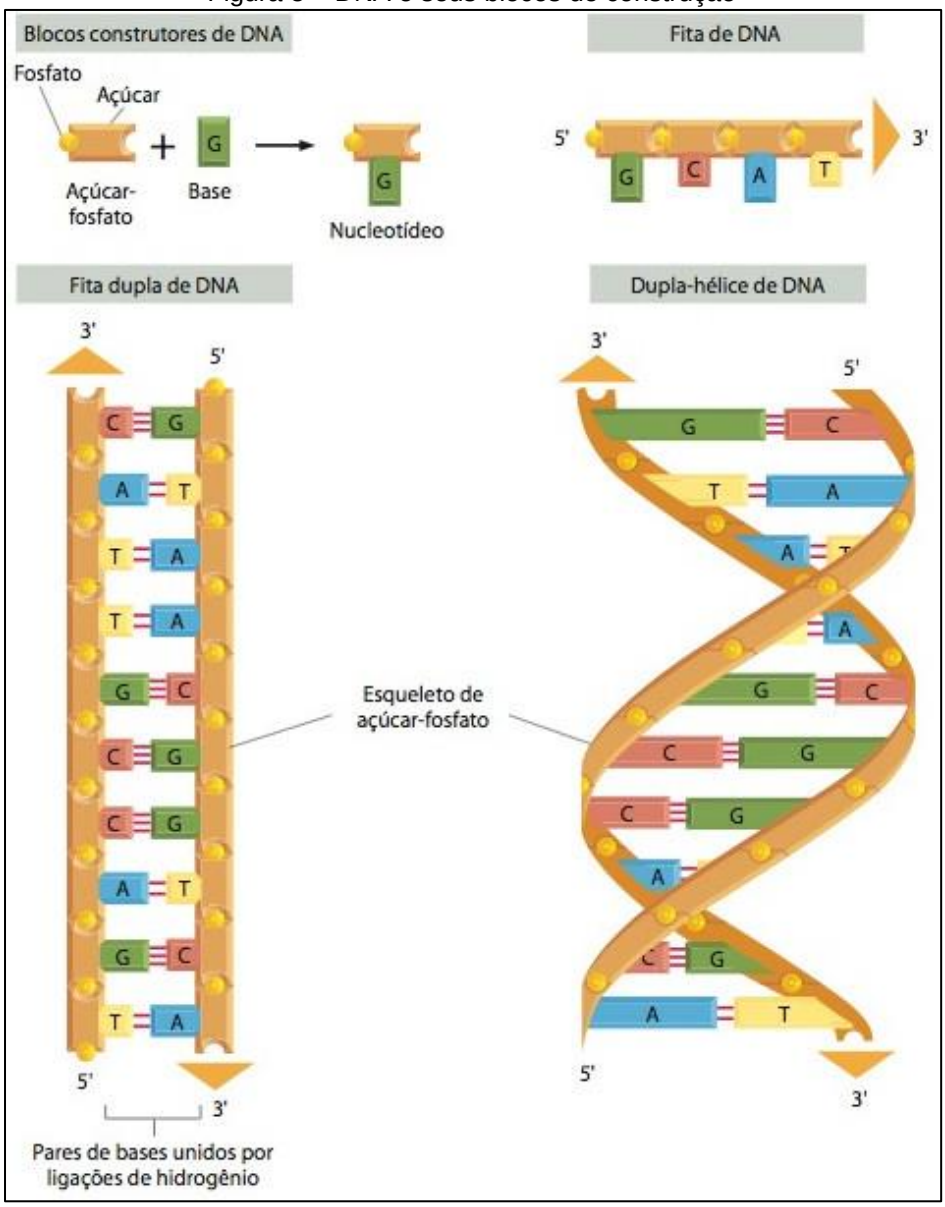
2.3 ÁCIDOS NUCLEICOS E PROTEÍNAS

De suma importância para os organismos vivos, os ácidos nucleicos são macromoléculas responsáveis por informar às células quais proteínas sintetizar. Estão armazenadas em unidades gênicas, nos cromossomos de uma célula. São dois os tipos de ácidos nucleicos: o ácido desoxirribonucleico (DNA) e o ácido ribonucleico (RNA) (ALBERTS et al., 2010).

2.3.1 DNA

O ácido desoxirribonucleico (ADN) é uma molécula formada por duas longas cadeias (ou fitas) polipeptídicas (açúcar-fosfato) com quatro tipos de nucleotídeos. O açúcar dos nucleotídeos, no DNA, é a desoxirribose. As bases nitrogenadas podem ser quatro: as purinas – adenina (A) e guanina (G) – e as pirimidinas – citosina (C) e timina (T). As bases dos nucleotídeos se conectam, umas com as outras, por ligações de hidrogênio, mantendo as cadeias unidas, como podemos ver na Figura 5 (SADAVA et al., 2009; ALBERTS et al., 2010).

Figura 5 – DNA e seus blocos de construção



Fonte: Alberts et al. (2010, p. 198).

A estrutura DNA, Figura 5, é brevemente descrita por Alberts et al. como um:

> [...] composto de quatro tipos de nucleotídeos, ligados covalentemente, formando uma cadeia polinucleotídica (uma fita de DNA), com um esqueleto de açúcar-fosfato a partir do qual as bases (A, C, G e T) se estendem. Uma molécula de DNA é composta de duas fitas de DNA unidas por ligações de hidrogênio entre as bases pareadas. As setas nas extremidades das fitas de DNA indicam as polaridades das fitas, que são antiparalelas entre si na molécula de DNA. [...] à esquerda da figura, o DNA está mostrado de forma plana; na realidade, ele é torcido formando uma dupla-hélice [...] (2010, p. 198).

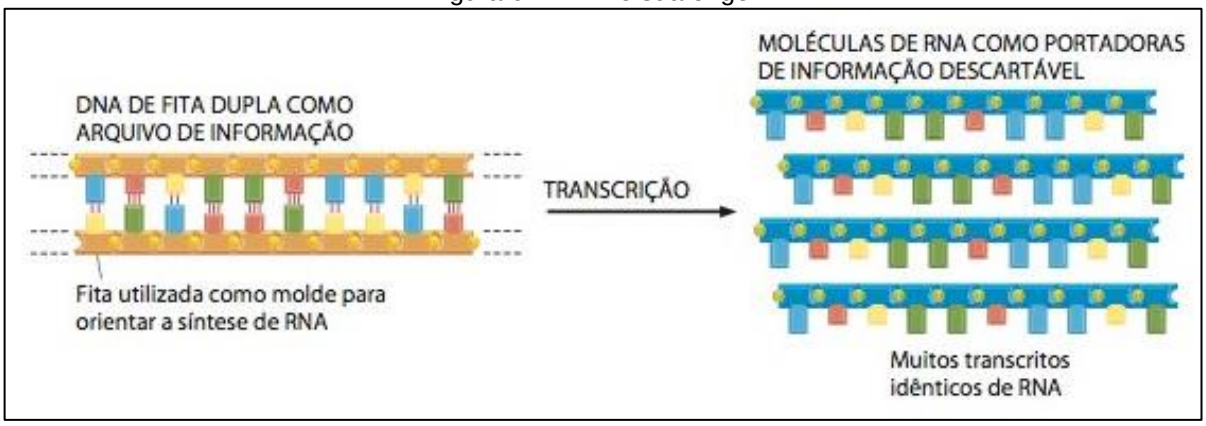
2.3.2 RNA

O ácido ribonucleico (ARN) é um polipeptídio como o DNA, mas não igual pois ele se difere em três aspectos:

a) possui apenas uma cadeia polipeptídica;
b) o açúcar encontrado é a ribose, ao contrário do DNA que é a desoxirribose;
c) contém as mesmas bases – adenina, guanina e citosina – que o DNA, com exceção à timina. No lugar dessa encontramos a uracila (U) que é muito similar quimicamente (BROWN, 2009; SADAVA et al., 2009).

Na Figura 6 é possível verificar as fitas de RNA, que são obtidas através de um DNA, o qual servirá de molde para o processo de transcrição que irá originar várias moléculas de RNA com funções distintas (ALBERTS et al., 2010).

Figura 6 – RNA e sua origem.



Fonte: Alberts et al. (2010, p. 5).

As diferentes classes, ou fitas, de RNA são classificadas conforme sua função e localização na célula. As principais classes são:

d) mRNA (RNA mensageiro): envia informações para o citoplasma, local onde ocorre a síntese de proteínas, que estavam contidas no DNA;
e) rRNA (RNA ribossômico): é parte do ribossomo da célula, local onde ocorre a síntese de proteínas;
f) tRNA (RNA transportador): transporta aminoácidos necessários para os ribossomos (ALBERTS et al., 2010; SANDERS; BOWMAN, 2014; ZAHA; FERREIRA; PASSAGLIA, 2014).

Os RNAs são dívidos em dois grupos: os RNAs codificantes – que possuem a informação para síntese de proteínas – e os RNAs não-codificantes – que não possuem essa informação como, por exemplo, o rRNA e o tRAN (ALBERTS et al., 2010).

2.3.3 Proteínas

As proteínas são moléculas não-ramificadas formadas por longos polímeros de aminoácidos. Assim como no DNA e no RNA, ela possui sua informação de forma linear e sequencial. Com exceção à água, a proteína é a maior parte da massa de uma célula (ALBERTS et al., 2010; COX; DOUDNA; O'DONNEL, 2012).

A proteína é resultado do processo de tradução do RNA. Nesse contexto, é a representação das expressões genéticas que determinará a composição de aminoácidos, os quais formarão a proteína (SADAVA et al., 2009; ZAHA; FERREIRA; PASSAGLIA, 2014).

Suas funções dentro da célula, ou entre células, abrangem o suporte estrutural, catálise de reações químicas, transporte de moléculas pelas membranas celulares, proteção, envio e recebimento de informações entre células, regulação e movimento (SADAVA et al., 2009; COX; DOUDNA; O'DONNEL, 2012).

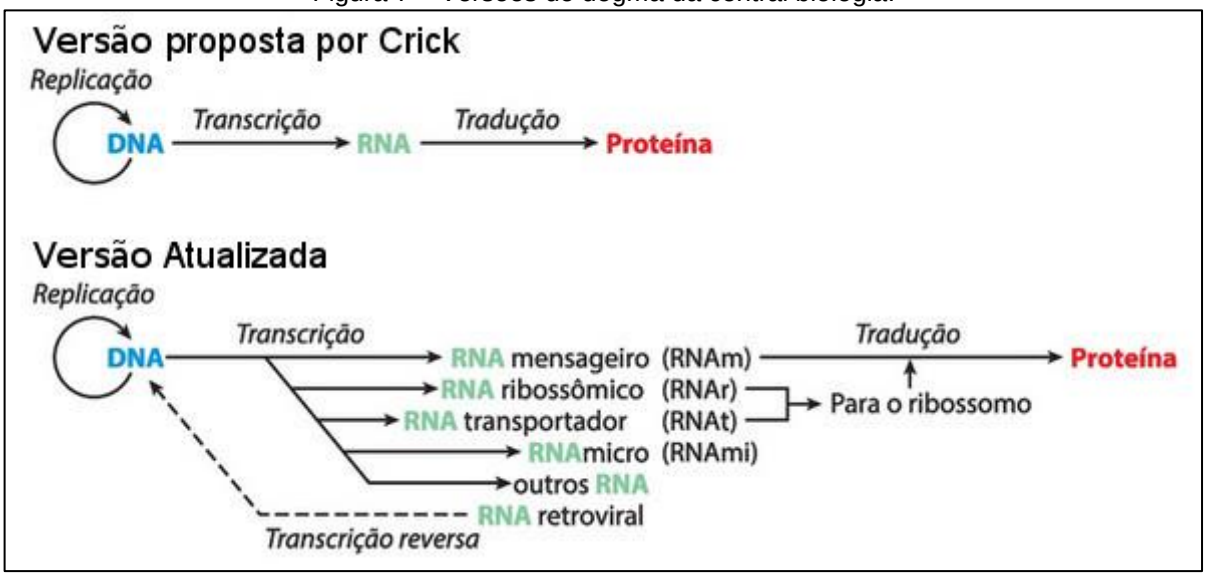
2.4 DOGMA CENTRAL DA BIOLOGIA MOLECULAR

O dogma central da biologia molecular é o ponto fundamental da doutrina da biologia molecular. Esse dogma descreve o fluxo da informação, ou seja, como obter proteínas a partir de moléculas de DNA.

Originalmente proposto por Francis Crick (1958) de forma bastante precisa, ao logo dos anos, as novas descobertas atualizaram o fluxo. Apesar das inovações, o modelo permanece com a estrutura original (COX; DOUDNA; O'DONNEL, 2012; SANDERS; BOWMAN, 2014).

O dogma pode ser melhor compreendido na Figura 7, onde compara-se a primeira versão com a atual (SANDERS; BOWMAN, 2014).

Figura 7 – Versões do dogma da central biologia.



Fonte: Sanders; Bowman (2014, p. 10).

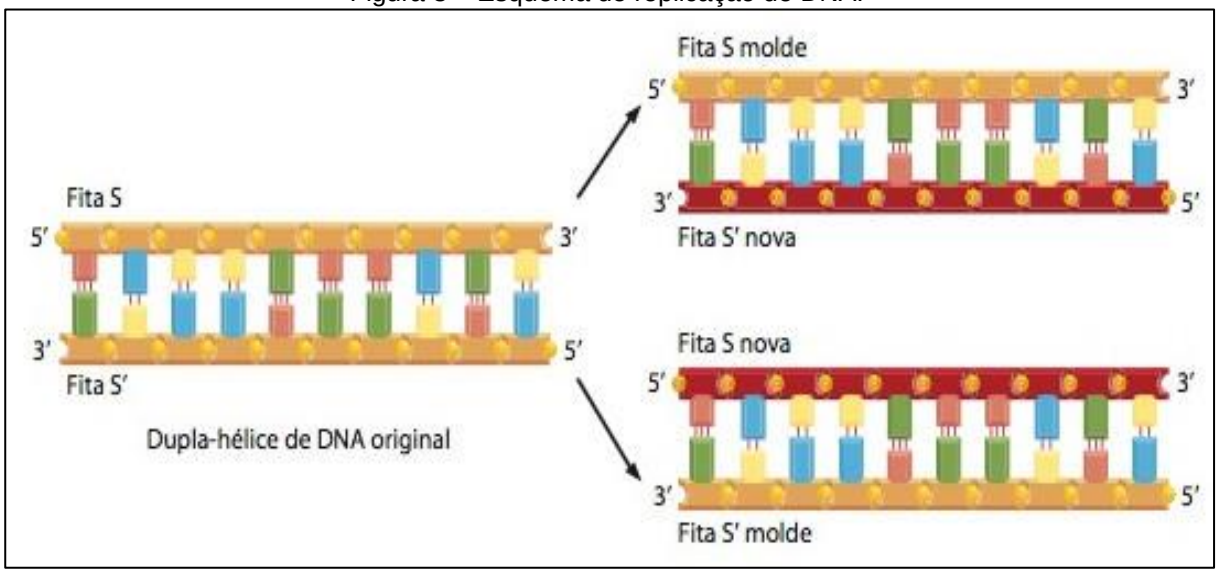
O fluxo principal (proposto por Crick, 1958) deixa claro que não é possível obter proteínas diretamente do DNA, havendo, dentro do fluxo, um único sentido para sintetizar proteínas. Por esse motivo, o tema passou a ter importante papel dentro da biologia (SADAVA et al., 2009; COX; DOUDNA; O'DONNEL, 2012; SANDERS; BOWMAN, 2014; VERLI, 2014).

Nos tópicos a seguir, serão descritos os processos de replicação de DNA, transcrição de DNA para RNA e a tradução de RNA para proteínas, pertencentes ao fluxo principal.

2.4.1 Replicação

A divisão celular, presente em todos os seres vivos, ocorre quando uma célula pai se divide originando células filhas. No início desse processo verifica-se a replicação de DNA, quando todo o genoma deverá ser copiado para os herdeiros. As novas cadeias de DNA são iguais à original, conforme se verifica na Figura 8 (BROWN, 2009; ZAHA et al., 2014).

Figura 8 – Esquema de replicação de DNA.

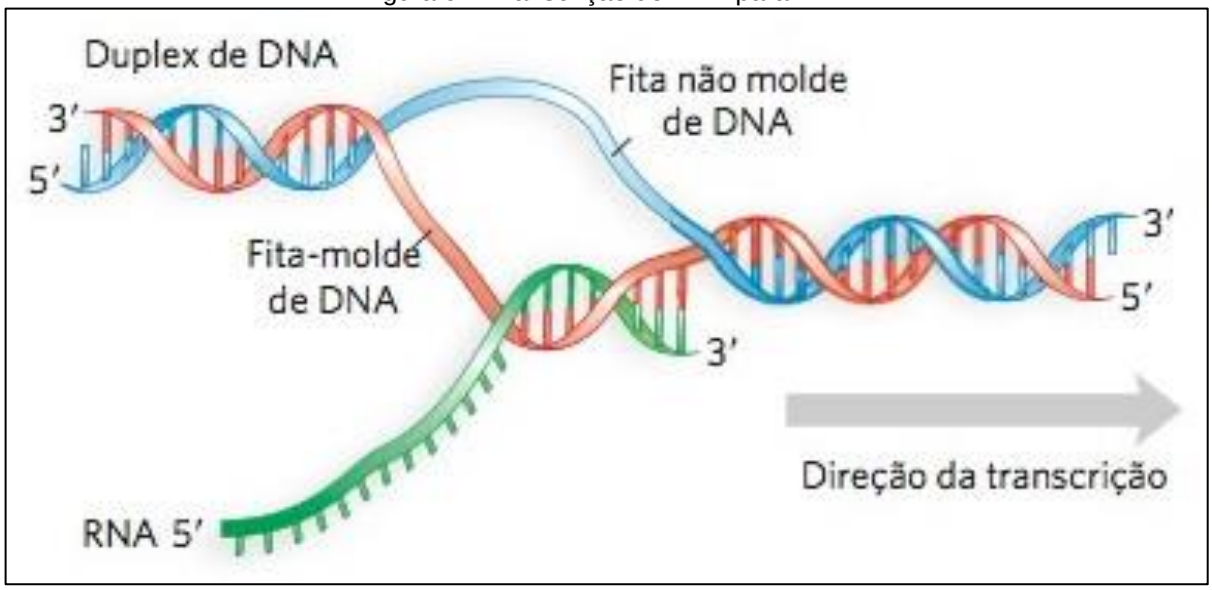


Fonte: Alberts et al. (2010, p. 266).

2.4.2 Transcrição

Conforme Cox, Doudna e O'Donnel (2012, p. 516), a transcrição é “a produção enzimática de uma fita exatamente complementar de RNA de um DNA-molde”. Ou seja, é o processo que sintetizará os RNAs mencionados nesse capitulo. O processo de transcrição está ilustrado na Figura 9.

Figura 9 – Transcrição do DNA para RNA



Fonte: Cox; Doudna; O'Donnel (2012, p. 516).

O processo possui três fases:

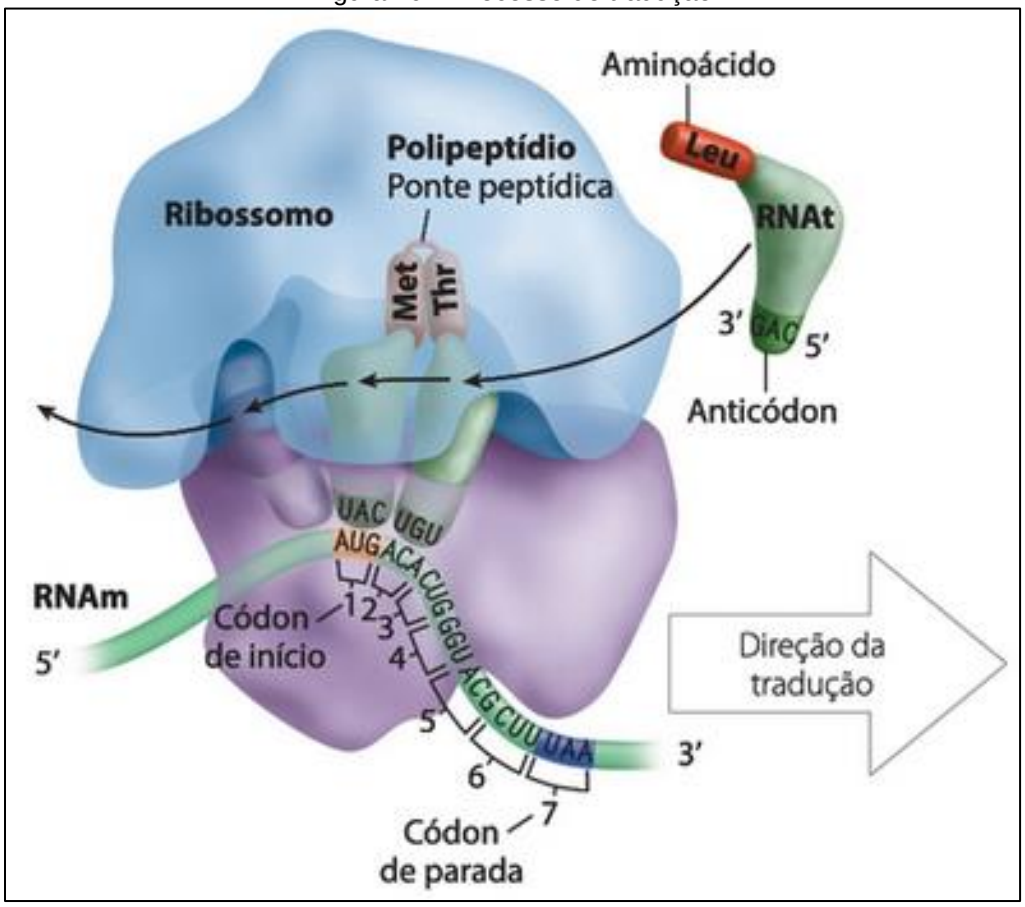
a) iniciação: sequências específicas do DNA (promotores) informam aonde iniciar a cópia, qual fita usar e a direção;
b) alongamento: o DNA é desenrolado para poder sintetizar o RNA;
c) terminação: sequências específicas do DNA indicam o ponto aonde termina o processo e então é liberada a fita-molde (SADAVA et al., 2009; COX; DOUDNA; O'DONNEL, 2012).

Para cada gene somente uma fita da dupla-hélice será transcrita, a fita molde. A outra fita, não-molde, não será transcrita por esse gene, mas pode servir de fita molde para outro gene (SADAVA et al., 2009).

2.4.3 Tradução

Esse processo acontece no ribossomo, onde o mRNA se conecta com seus conjuntos de códons. O códon de início determina onde começar a tradução, percorrendo a fita de mRNA no sentido 5’ → 3’. A tradução (Figura 10) é o processo que sintetizará proteínas a partir dos RNAs (SANDERS; BOWMAN, 2014).

Figura 10 – Processo de tradução



Fonte: Sanders; Bowman (2014, p. 12).

O ribossomo está ligado à fita de mRNA através de suas pontes peptídicas. Essa ligação monta os aminoácidos conforme a ordem em que aparecem na fita de mRNA. A tradução termina quando é encontrado o códon de parada na fita O tRNA faz o transporte de aminoácidos para o ribossomo. Além disso, ele lê o mRNA para definir quais aminoácidos deverão ser entregues para o ribossomo catalisar a proteína Uma molécula de DNA transcreve muitas sequências de RNAm, sendo que cada sequência corresponde a um gene que pode traduzir diferentes proteínas (SANDERS; BOWMAN, 2014; GOWDAK; MATTOS; PEZZI, 2013).

2.5 REGIÕES INTERGÊNICAS E PROMOTORES

A região intergênica é uma parte do genoma também conhecida por região não-codificante. Ela é toda sequência genômica que está localizada entre duas sequências

codificantes. É dentro dessa região que estão localizados os promotores (ALBERTS et al., 2010; AVILA E SILVA, 2011; ZAHA; LESSA, 2012; FERREIRA; PASSAGLIA, 2014).

Promotores são sequências especificas do DNA que atuam como reguladores no processo de transcrição. O promotor informa onde começar a transcrição, qual dupla-hélice de DNA será a fita-molde e a direção que deve seguir. A região promotora está localizada anteriormente a uma região codificante (SADAVA et al., 2009; AVILA E SILVA, 2011; ZAHA et al., 2014).

2.6 BIOINFORMÁTICA

A complexidade e a quantidade de informações envolvidas nos processos do dogma central da biologia inviabilizam o ser humano de geri-las. A bactéria *E. coli*, por exemplo, possui, aproximadamente, 4.300 genes, gerando um genoma (volume de dados) de 4,6MB. Esse montante de dados somente poderá ser processado por uma máquina (VERLI, 2014; ZAHA; FERREIRA; PASSAGLIA, 2014).

A bioinformática é uma ciência que utiliza técnicas computacionais para processar informações da área da biologia. O cenário atual tem sido muito favorável à bioinformática, desde avanços tecnológicos para melhoria e o baixo custo do hardware, até as diversas ferramentas de software e suas constantes evoluções para melhor atender às necessidades dos usuários (VERLI, 2014).

Grandes projetos como, por exemplo, o Projeto Genoma Humano e os sequenciadores automáticos de DNA, têm impulsionado a área gerando uma necessidade massiva e exponencial de recursos computacionais que permitam armazenar, organizar, processar e refinar informações. (VERLI, 2014; ZAHA; FERREIRA; PASSAGLIA, 2014).

Nesse capítulo, serão analisados os principais bancos de dados e portais de biologia molecular aonde concentram-se essas bases.

2.6.1 Banco de dados de biologia molecular

Um Banco de Dados (BD), segundo Lesk (2008), é um conjunto de arquivos com informações, organizados logicamente e com meios de acesso a tais informações. Nesse contexto, um Banco de Dados de Biologia Molecular (BDBM) segue o mesmo conceito de um BD, mas aplicado à biologia molecular, contendo informações como, por exemplo, sequências de ácidos nucleicos e de proteínas, estruturas e funções de macromoléculas, padrões de expressão, entre outros.

Os BDBMs, segundo Biotecnologia (2002), podem ser classificados em dois grupos: os primários – que armazenam dados não processados – e os secundários – criados a partir dos primários, tendo algum processamento ou análise realizada.

2.6.2 Portais de bioinformática

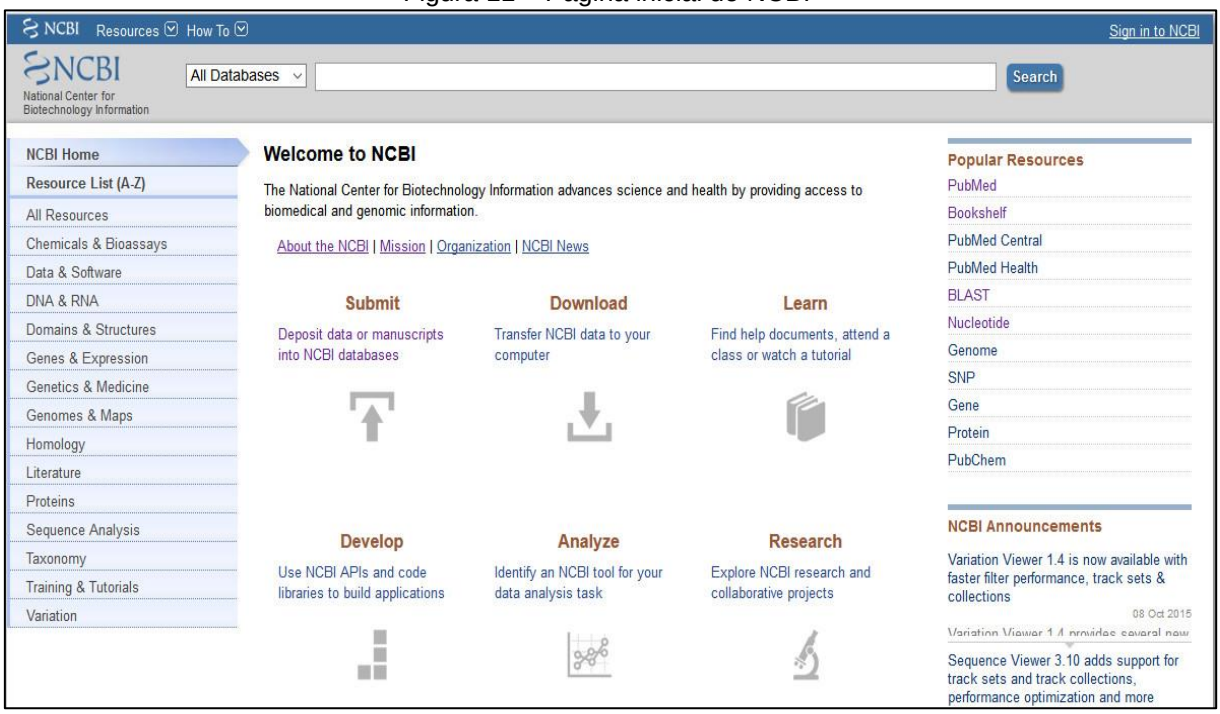
Portal é um site ou conjunto de sites da *Internet* que, através de hipertextos, disponibiliza serviços e informações sobre um ou mais assuntos (MARÇULA; BENINI FILHO, 2013). Os portais de bioinformática não são diferentes, pois disponibilizam recursos e informações através da mesma tecnologia tendo como assunto central a biologia molecular. Tornar o acesso público se faz necessário para que os dados sejam facilmente acessados e não se tornem um cemitério de informações (LESK, 2008).

2.6.2.1 NCBI

Um exemplo de portal é o *National Center for Biotechnology Information* (NCBI) que foi criado em 1988, pelo então falecido senador Claude Pepper. Constituído como uma subdivisão do NLM (*National Library of Medicine*) no NIH (*National Institutes of Health*), para atender às necessidades de métodos computacionais para realização de investigações biomédicas. A coletividade de ferramentas e recursos do NIH formam o maior centro de pesquisa biomédica do mundo[2] (NCBI, 2015).

O NCBI (Figura 11) é um recurso público que tem como missão “desenvolver novas tecnologias de informação para ajudar na compreensão dos processos moleculares e genéticos que controlam a saúde e as doenças” [3] (NCBI,2015).

Figura 11 – Página inicial do NCBI



Fonte: NCBI[4] (2015).

[2] Disponível em: <http://www.ncbi.nlm.nih.gov/home/about/mission.shtml>. Acessado em 24/10/2015.

[3] Disponível em: <http://www.ncbi.nlm.nih.gov/home/about/mission.shtml>. Acessado em 24/10/2015.

[4] Disponível em: <http://www.ncbi.nlm.nih.gov>. Acessada em: 24/10/2015.

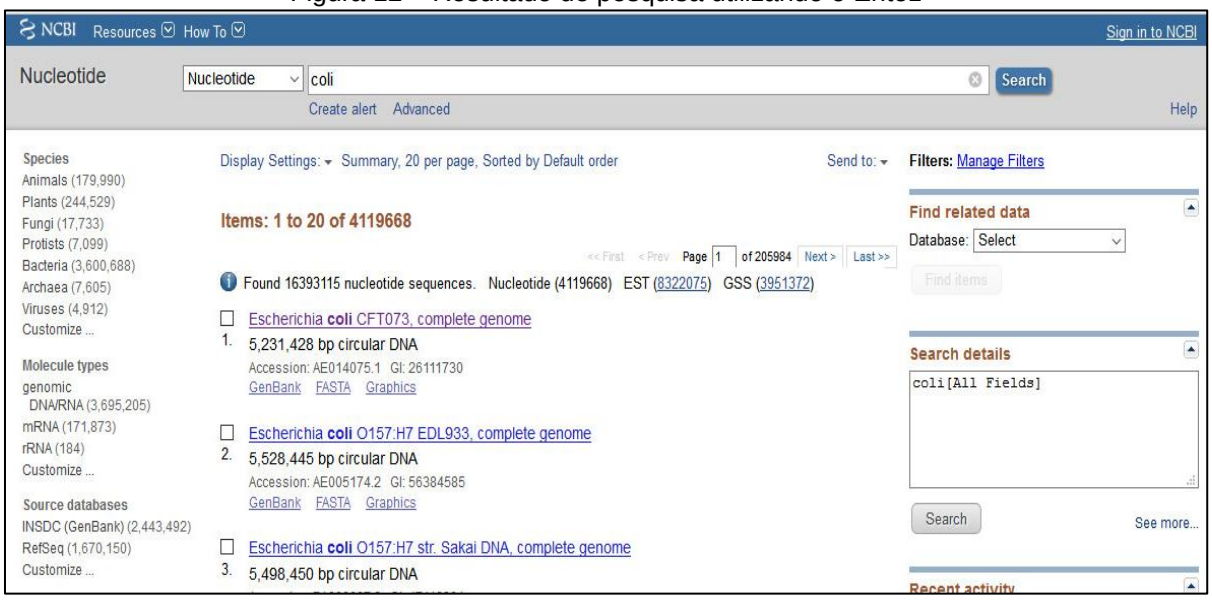
O NCBI possui inúmeros serviços importantes, tais como o *PubMed* – base com mais de 25 milhões de citações nas literaturas biomédicas[5] –, o *BookShelf* – uma coleção de livros, artigos, periódicos e outros recursos acadêmicos da biologia, medicina e ciência da vida[6] (NCBI, 2015).

O *GenBank*, uma base de dados de sequências genéticas, é um dos serviços mais conhecidos do NCBI, que faz parte de uma colaboração internacional – *International Nucleotide Sequence Database Collaboration* (INSDC), formada por DDBJ (*DNA DataBank Japan*), EMBL (*European Molecular Biology Laboratory*) e *GenBank* da NCBI – que trocam dados diariamente[7]. Atualmente, o *GenBank* possui mais de 199 bilhões de bases e 187 milhões de sequencias[8] (NCBI, 2015).

Para acessar todas essas bases, o NCBI disponibiliza diversas ferramentas que se distinguem não só pela interface, mas pelo método que irão utilizar para trazer o resultado desejado. A mais utilizada dentre elas é o Entrez, tido como o sistema primário de busca e recuperação textual (NCBI, 2015).

A ferramenta interage com a base de dados, a literatura biomédica do *Pubmed* e outros 39 bancos de dados de literaturas moleculares[9]. A Figura 12 mostra uma pesquisa simples utilizando o Entrez e o termo “coli” como parâmetro (NBCI, 2015).

Figura 12 – Resultado de pesquisa utilizando o Entez

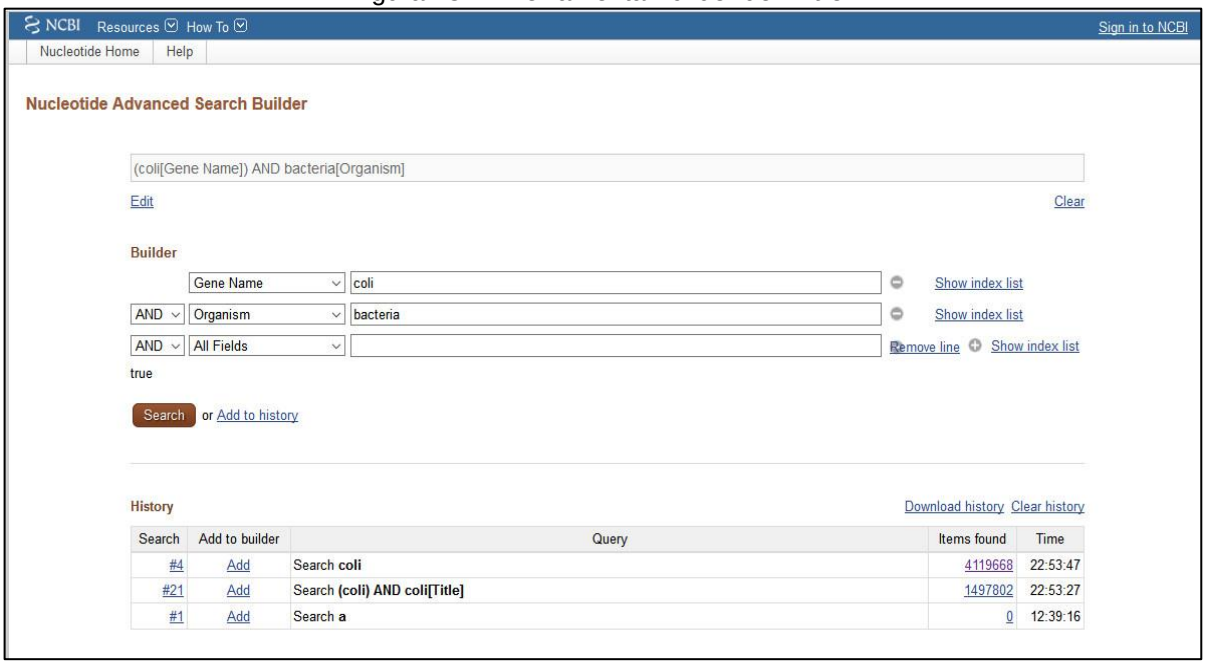


Fonte: NCBI[10] (2015).

[5] Disponível em: <http://www.ncbi.nlm.nih.gov/books/NBK3833>. Acessada em: 24/10/2015.
[6] Disponível em: <http://www.ncbi.nlm.nih.gov/books/NBK3833>. Acessada em: 24/10/2015.
[7] Disponível em: <http://www.ncbi.nlm.nih.gov/genbank>. Acessada em: 24/10/2015.
[8] Disponível em: <http://www.ncbi.nlm.nih.gov/genbank/statistics>. Acessada em: 11/10/2015.
[9] Disponível em: <http://www.ncbi.nlm.nih.gov/books/NBK3837>. Acessada em: 24/10/2015.
[10] Disponível em: <http://www.ncbi.nlm.nih.gov/nuccore/?term=coli>. Acessada em: 11/10/2015.

Na Figura 13, pode-se ver uma tela com mais recursos, permitindo inserir mais parâmetros à pesquisa, gerando um resultado mais preciso (NBCI, 2015).

Figura 13 – A ferramenta Builder do Entrez



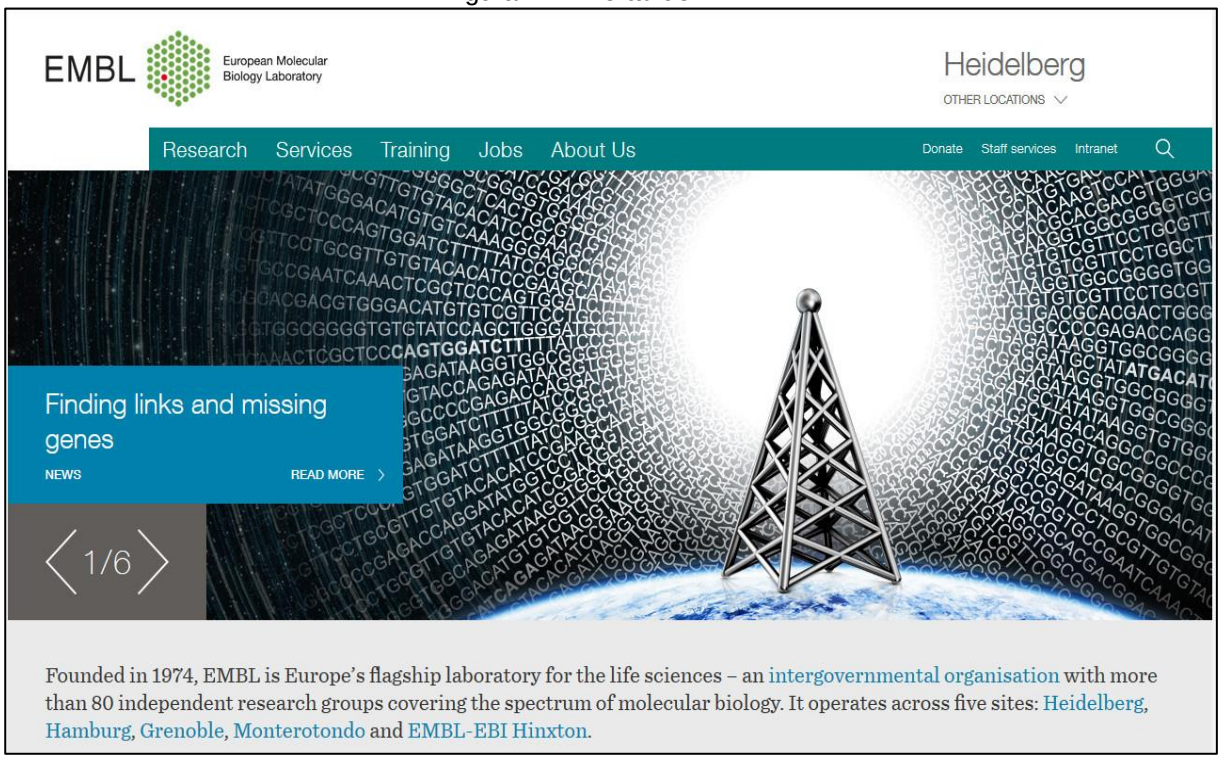
Fonte: NCBI[11] (2015).

2.6.2.2 EMBL

O *European Molecular Biology Laboratory* (EMBL), foi idealizado em 1962 pelos cientistas Leo Szilárd, James D. Watson e John C. Kendrew. A ideia era criar um centro de pesquisas para equilibrar forças no campo da biologia molecular, até então dominada pelos EUA (Estados Unidos da América), promovendo a área por toda a Europa. A sede do EMBL fica em Heidelberg, na Alemanha, mas possui outros quatro laboratórios espalhados pela Europa. A Figura 14 ilustra a página inicial do portal do EMBL[12] (EMBL, 2015).

[11] Disponível em: <http://www.ncbi.nlm.nih.gov/nuccore/advanced>. Acessada em: 11/10/2015.
[12] Disponível em: <http://www.embl.de/aboutus/general_information/history/index.html>. Acessada em: 12/10/2015.

Figura 14 – Portal do EMBL



Fonte: EMBL[13] (2015).

Atualmente, é considerado o principal laboratório da Europa no campo de pesquisas em biologia molecular[14]. Isso se tornou possível devido às suas cinco principais missões:

a) **Pesquisa Básica em Biologia Molecular:** a compreensão fundamental dos processos biológicos básicos em organismos modelo. Isso é possível através de uma estrutura interdisciplinar integrada para estudar e compreender os fenômenos e sistemas biológicos em toda a sua complexidade;

b) **Tecnologia e Instrumentalização:** desenvolvimento de instrumentos, tecnologia, softwares e bancos de dados para as ciências da vida;

c) **Instalações e Serviços:** além do portal e das bases de dados fornecidas pelo EMBL, o centro disponibiliza aos pesquisadores acesso a inúmeros serviços, instalações, laboratórios e equipamentos para estudo e realização de experimentos;

[13] Disponível em: <http://www.embl.de/index.php>. Acessada em: 12/10/2015.
[14] Disponível em: <http://www.embl.de/aboutus/general_information/history/index.html>. Acessada em: 12/10/2015.

d) **Ensino e Formação:** visando o aperfeiçoamento, o centro dispõe de formação para PhDs (*Doctor of Philosophy*), pós-doutorados, orientação a jovens professores, colaboração em pesquisas de terceiros, cursos de formação, workshops, conferências e simpósios reconhecidos mundialmente;

e) **Transferência de Tecnologia:** ativamente envolvida no desenvolvimento de suas descobertas em benefício da sociedade. Auxilia na identificação, proteção e comercialização da propriedade intelectual desenvolvida no EMBL e parceiros. Apoia e ajuda a acelerar a transferência de tecnologias inovadoras de suas pesquisas para a indústria[15] (EMBL, 2015).

O *European Nucleotide Archive* (ENA) é um dos serviços desenvolvidos e mantidos pelo EMBL-EBI sob a orientação do INSDC (*International Advisory Committee*) e um conselho (*Scientific Advisory Board*). Esse serviço recebe e apresenta informações relativas aos trabalhos experimentais que são baseados em torno de sequências de nucleotídeos[16] (EMBL, 2015).

A base é abastecida e compartilhada de várias formas com parceiros como, por exemplo, o INSDC. A colaboração envolve rotinas que vão desde o envio e recebimento de dados brutos até sequências montadas e anotações sobre sequenciamento em pequena escala. O ENA registra essas informações em um modelo de dados que abrange informações de entrada, dados da máquina e de interpretações[17].Atualmente, o banco de dados possui mais de 641 milhões de sequências e 1,4 bilhões de bases[18] (EMBL, 2015).

O ENA dispõe do ENA Browser para realização de pesquisa e recuperação das informações em suas bases de forma interativa e programática. O ENA Browser é composto por quatro ferramentas: a *Free Text Search*, a *Advanced Search*, a *Sequence Similarity Search* e o *Bulk Data Download*.

A ***Free Text Search*** é uma ferramenta para pesquisa livre por texto. O campo para realizar a pesquisa está disponibilizado em cabeçalhos em várias sessões do site EMBL-EBI[19]. Na Figura 15 pode-se ver o campo de pesquisa com o termo *“coli”*

[15] Disponível em: <http://www.embl.de/aboutus/general_information/mission/index.html>. Acessada em: 12/10/2015.
[16] Disponível em: <http://www.ebi.ac.uk/ena/about>. Acessada em: 12/10/2015.
[17] Disponível em: < http://www.ebi.ac.uk/ena/about/>. Acessada em: 12/10/2015.
[18] Disponível em: <http://www.ebi.ac.uk/ena/about/statistics>. Acessada em: 12/10/2015.
[19] Disponível em: <http://www.ebi.ac.uk/ena/browse>. Acessada em: 21/11/2015.

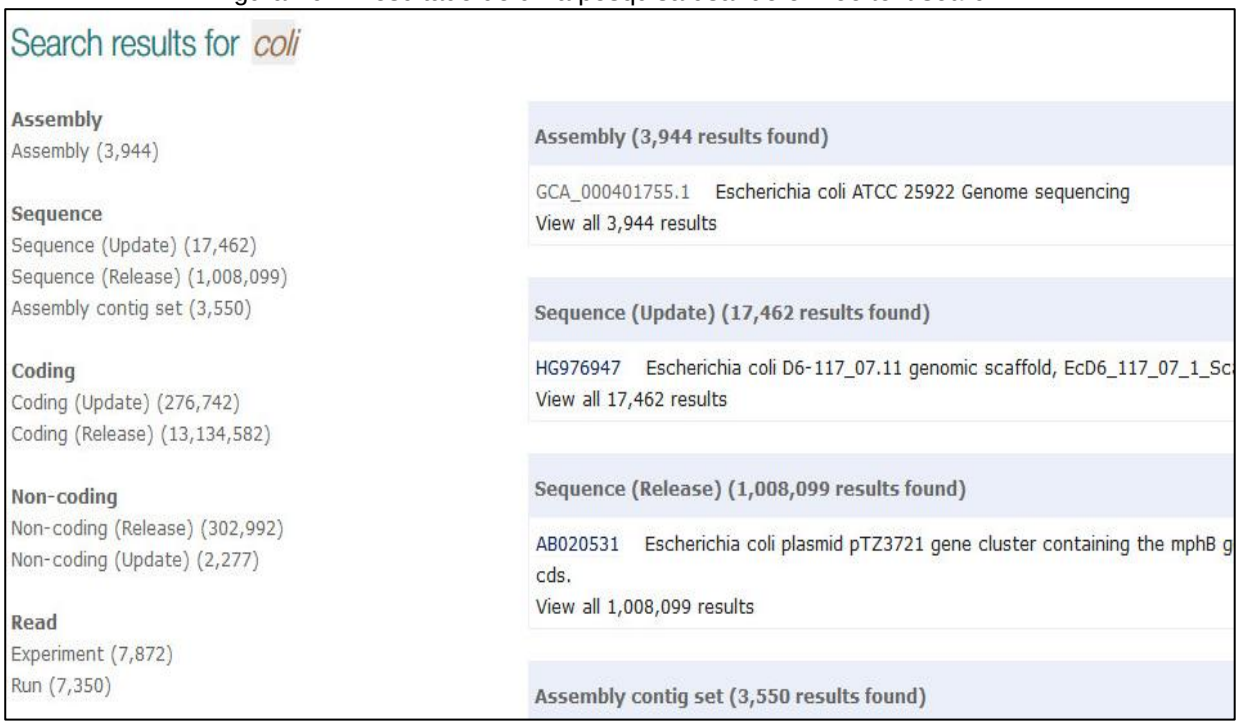
digitado. Nele também se encontra, abaixo do botão *Search*, o *link* de acesso às ferramentas *Advanced Search* e *Sequence Similarity Search* (EMBL, 2015).

Figura 15 – EMA Free text search

Fonte: EMBL[20] (2015).

Abaixo, na Figura 16, pode-se ver o resultado obtido da pesquisa usando o termo *"coli"*. A tela está dívida em duas colunas, tendo ambas as mesmas informações e *links* para outras sessões separadas por categorias. O lado esquerdo difere por ter uma formatação específica e algumas linhas de texto com informações resumidas sobre o item em destaque.

Figura 16 – Resultado de uma pesquisa usando o *Free text search*

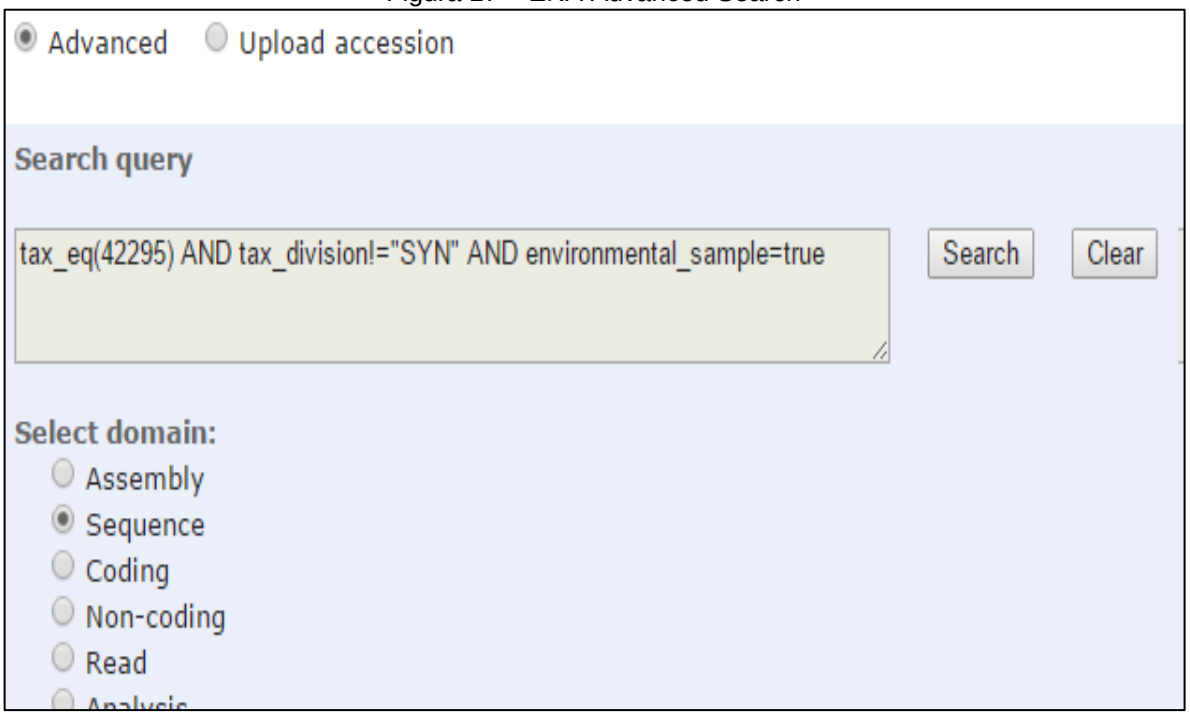


Fonte: EMBL[21] (2015).

[20] Disponível em: <http://www.ebi.ac.uk/ena/data/search?query=coli>. Acessada em: 21/11/2015.
[21] Disponível em: <http://www.ebi.ac.uk/ena/data/search?query=coli>. Acessada em: 21/11/2015.

A ***Advanced Search*** é um recurso similar ao *Free text search*, mas com inúmeras opções de filtros. Na Figura 17, pode-se ver a criação de uma *Search query* (instrução para pesquisa, linha de texto que representa a pesquisa que se desejasse realizar). Esse texto é gerado conforme os campos abaixo são selecionados e preenchidos. Somente um único domínio pode ser selecionado por pesquisa e, para cada domínio selecionado, parâmetros diferentes serão disponibilizados. Na Figura 17 está selecionado o domínio *Sequence (sequências),* que possui mais de 40 parâmetros diferentes (EMBL, 2015).

Figura 17 – ENA Advanced Search



Fonte: EMBL[22] (2015).

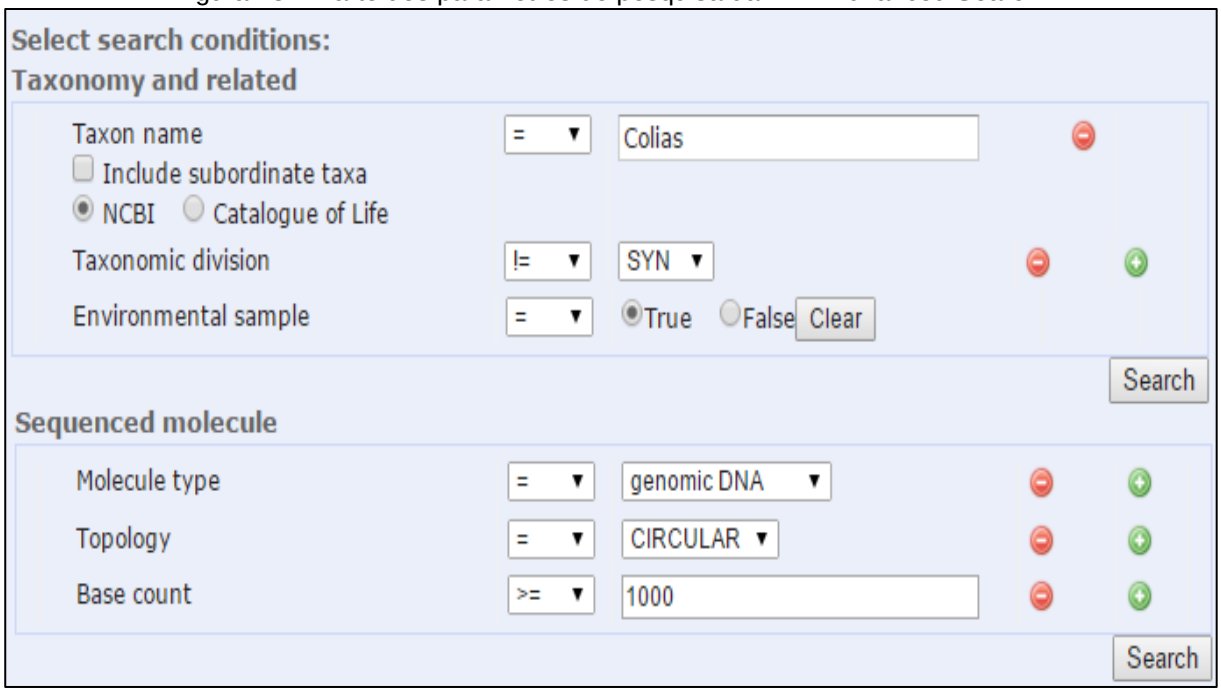
Na Figura 18, estão ilustrados os parâmetros que foram selecionados e formam a linha de pesquisa apresentada, anteriormente, na Figura 17. A Figura 18 foi manipulada, para poder mostrar esse conteúdo de forma legível. Ao utilizar a ferramenta, observa-se que:

a) os campos são pré-definidos para o domínio selecionado;
b) alguns campos de texto possuem o recurso de auto complemento;
c) a maioria dos campos possuem um pré-campo com operadores lógicos, possibilitando uma maior precisão do resultado;

[22] Disponível em: <http://www.ebi.ac.uk/ena/data/warehouse/search>. Acessada em: 21/11/2015.

d) os campos podem ser inclusos mais de uma vez ou removidos da pesquisa;

Figura 18 – Parte dos parâmetros de pesquisa da ENA Advanced Search



Fonte: Adaptado de EMBL[23] (2015).

A ***Sequence Similarity Search***[24], adicionada em março de 2015 como uma nova ferramenta de busca, permite ao usuário pesquisar informações a partir de uma sequência inteira ou parcial. O texto pode ser inserido no campo de pesquisa através da área de transferência (copiar e colar) ou fazendo o *upload* de um arquivo. Parâmetros para filtragem do resultando são bastantes variados, permitindo a seleção de um subconjunto de sequências, limitação por valores máximos e mínimos, escolha do programa (algoritmo) que será utilizado, entre outros[25] (EMBL, 2015).

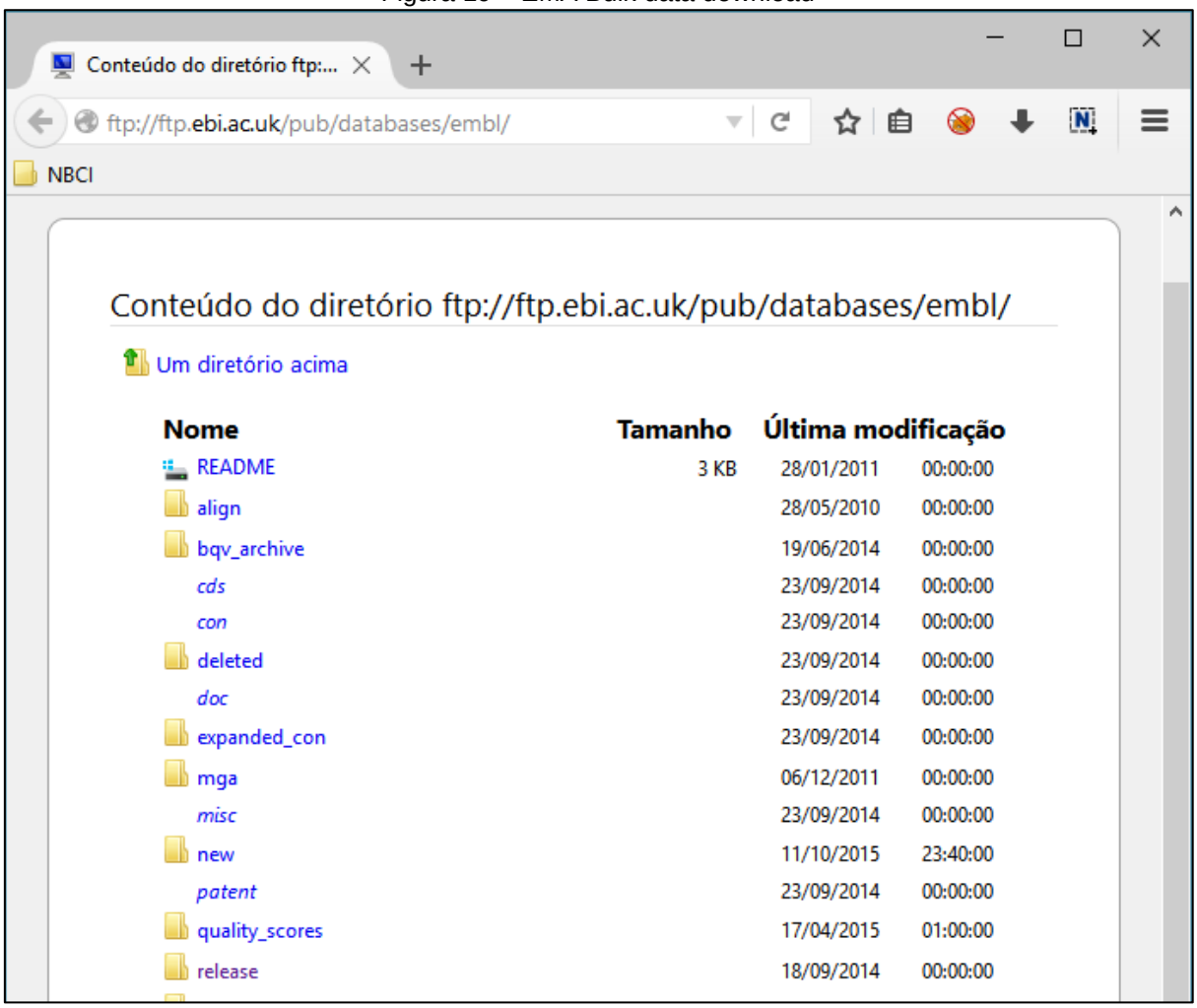
O ***Bulk Data Download***[26] é utilizado somente para *downloads* de informações em grande escala utilizando o protocolo FTP (*File Transfer Protocol*). As informações estão organizadas e uma estrutura de diretórios (pastas e subpastas) com arquivos dos textos, arquivos compactados, entre outros formatos (EMBL, 2015).

Requer dos utilizadores conhecimentos tanto sobre FTP como da estrutura de pastas criadas pelo EMBL-EBI. Na Figura 19 está ilustrada o acesso via FTP,

[23] Disponível em: <http://www.ebi.ac.uk/ena/data/warehouse/search>. Acessada em: 21/11/2015.
[24] Disponível em: <http://www.ebi.ac.uk/ena/data/sequence/search>. Acessada em: 12/10/2015.
[25] Disponível em: <http://www.ebi.ac.uk/ena/browse/sequence-search>. Acessada em: 21/11/2015.
[26] Disponível em: <http://www.ebi.ac.uk/ena/browse/download>. Acessada em: 21/11/2015.

utilizando o Mozilla Firefox como cliente FTP. A visualização e os recursos podem variar de um cliente de FTP para outro.

Figura 19 – EMA Bulk data download



Fonte: EMBL[27] (2015).

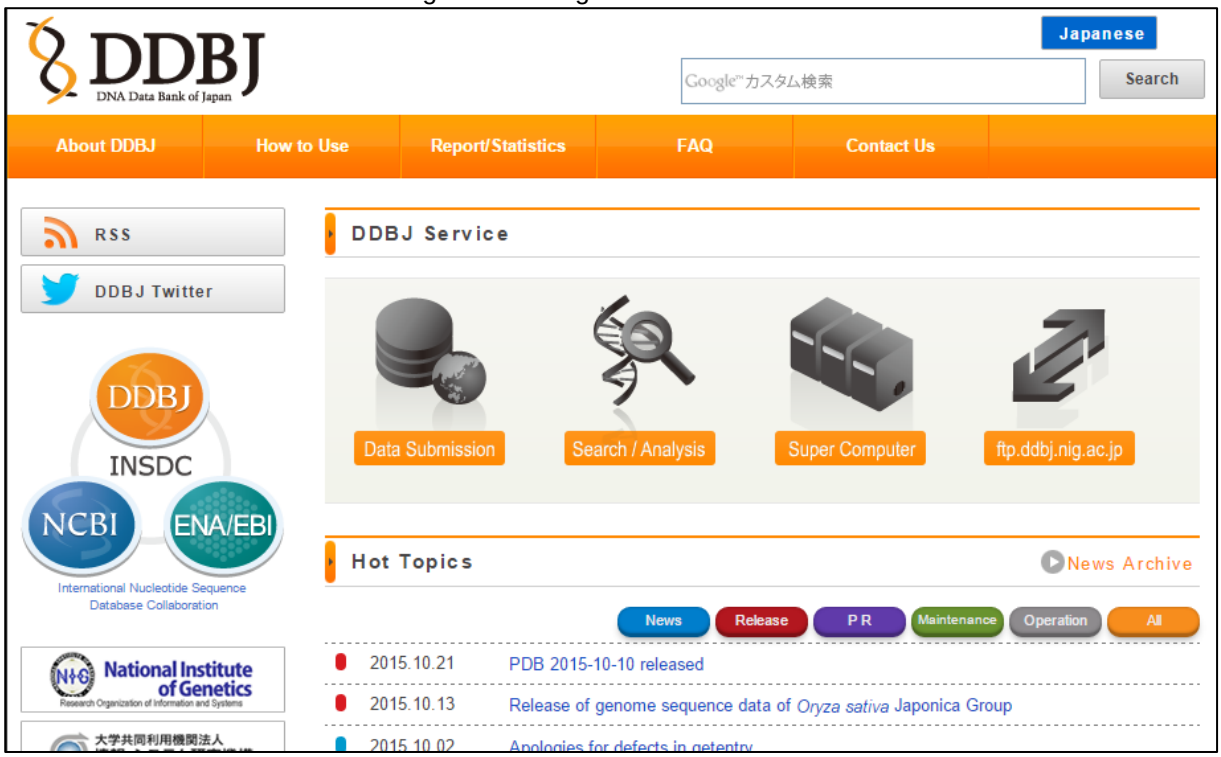
2.6.2.3 DDBJ

O *DNA Data Bank Japan* (DDBJ), criado em 1980, recolhe e armazena informações sobre sequências de nucleotídeos e fornece acesso a esses dados e sistemas para apoiar pesquisas relacionadas à ciência da vida. O objetivo principal do

[27] Disponível em: <ftp://ftp.ebi.ac.uk/pub/databases/embl/>. Acessada em: 12/10/2015.

DDBJ é melhorar a qualidade do INSDC, detalhando o quanto for possível as informações sobre os dados[28]. O site oficial está ilustrado na Figura 20 (DDBJ, 2015).

Figura 20 – Página inicial do DDBJ



Fonte: DDBJ[29] (2015).

Seus quatro principais serviços *on-line* são: o *Data Submission*, o *Search and Analysis*, o *Super Computer* e o *Archives*. Todos eles são acessíveis a partir da página inicial do DDBJ como pode-se ver na Figura 20 (DDBJ, 2015).

O ***Data Submission***[30] permite aos pesquisadores inserir dados de pesquisas na base de dados do DDBJ. O envio dos dados, ou submissão, pode ser feito através da página web ou, no caso de um grande volume de informações, através do MMS[31] (DDBJ, 2015).

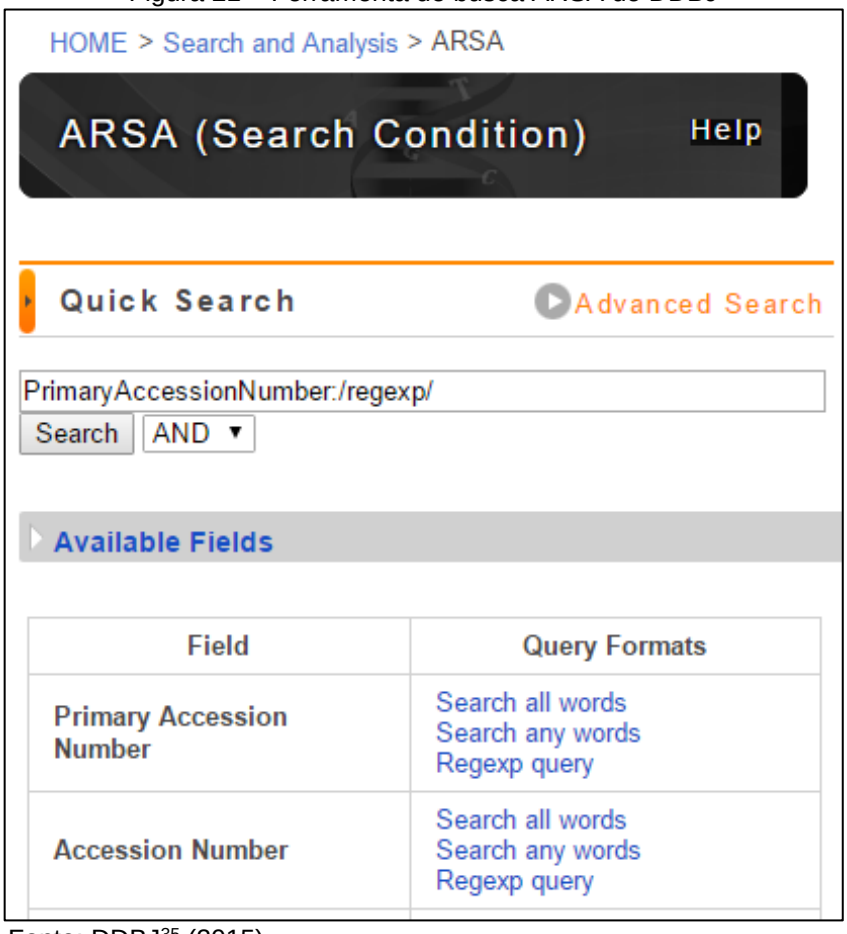
O ***Super Computer***[32], também conhecido por *NIG Supercomputer System*, são computadores e sistemas de grande porte, utilizados principalmente para a análise de genomas (DDBJ, 2015).

[28] Disponível em: <http://www.ddbj.nig.ac.jp/intro-e.html>. Acessada em: 24/10/2015.
[29] Disponível em: <http://www.ddbj.nig.ac.jp/index-e.html>. Acessada em: 24/10/2015.
[30] Disponível em: <http://www.ddbj.nig.ac.jp/submission_general-e.html>. Acessada em: 24/10/2015.
[31] Disponível em: <http://www.ddbj.nig.ac.jp/submission>. Acessada em: 24/10/2015.
[32] Disponível em: <http://sc.ddbj.nig.ac.jp/index.php/en/>. Acessada em: 24/10/2015.

O serviço ***Archives***[33], por sua vez, engloba diferentes ferramentas para acessar todas as informações contidas no DDBJ. A principal dessas ferramentas é o acesso pelo protocolo FTP, que permite fazer o *download* de informações nos formatos de arquivos .TXT e .XML (DDBJ, 2015).

O ***Search and Analysis***[34] consiste em diversas ferramentas que permitem a procura e a análise de informações dentro do banco de dados. Esse serviço compreende mais de vinte ferramentas distintas. Dentre elas, a ARSA ou *Search Condition*, ilustrada na Figura 21 (DDBJ, 2015).

Figura 21 – Ferramenta de busca ARSA do DDBJ



Fonte: DDBJ[35] (2015)

A ferramenta permite a procura por um ou mais termos. A inclusão (AND) ou a diferenciação (OR) pode ser selecionada abaixo do campo onde são digitados os termos da pesquisa (DDBJ, 2015).

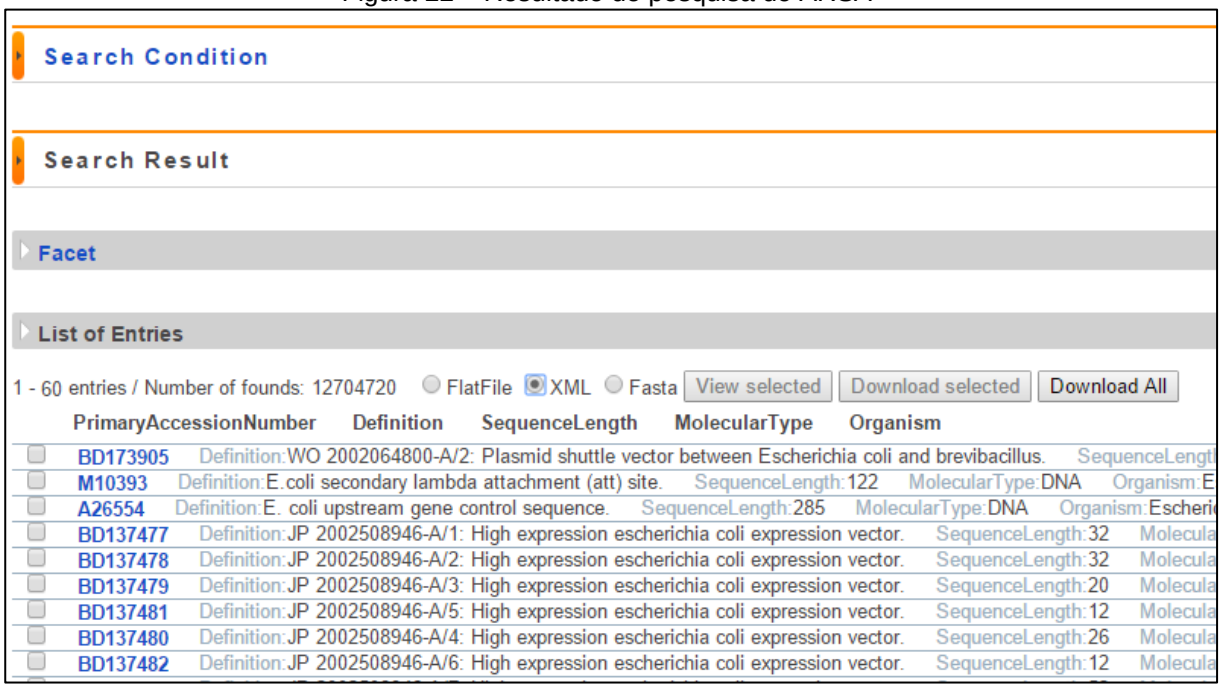
[33] Disponível em: <http://www.ddbj.nig.ac.jp/ftp_soap-e.html>. Acessada em: 24/10/2015.
[34] Disponível em: <http://www.ddbj.nig.ac.jp/searches-e.html>. Acessada em: 24/10/2015.
[35] Disponível em: <http://ddbj.nig.ac.jp/arsa/>. Acessada em: 21/11/2015.

Outro recurso disponibilizado na tela é o *Available Fields*, que consiste em textos pré-determinados (*query formats*) para a localização mais especifica de uma determina informação. Na Figura 21, pode-se ver que o campo foi preenchido com um texto que foi obtida automaticamente ao clicar no *link "Regexp query"*. Nesse exemplo, o usuário somente precisa substituir o trecho "/regexp/" pelo valor que deseja procurar dentro da categoria *Accesion Number*, que originou o texto (DDBJ, 2015).

Selecionando a opção *Advanced Search*[36], serão disponibilizados mais campos para parametrizar a pesquisa, ou seja, filtrá-la. Também dispõe de opções de ordenação do resultado e quaisquer campos (*fields*) (DDBJ, 2015).

O resultado de uma pesquisa, utilizando o termo "coli", realizado pelo ARSA está ilustrado na Figura 22. Cada linha contém uma informação distinta presente no banco de dados, sendo que cada informação contém o temo "coli" em algum lugar do registro (DDBJ, 2015).

Figura 22 – Resultado de pesquisa do ARSA



Fonte: DDBJ[37] (2015).

A tela permite o *download* de todas as informações, mesmo as que não estão sendo exibidas. Também é possível fazer o *download* de alguns itens através de seleção, utilizando o *checkbox* existente no início de cada linha. Os formatos de

[36] Disponível em: <http://ddbj.nig.ac.jp/arsa/advanced_search?lang=en>. Acessada em: 25/10/2015.
[37] Disponível em: <http://ddbj.nig.ac.jp/arsa/search?lang=en&cond=quick_search&query=coli>. Acessada em: 21/11/2015.

arquivos para *download* são TXT, XML e FASTA. Quando selecionado mais de um objeto para *download*, cada item será disponibilizado no formato selecionado dentro de um único arquivo compactado no formato GZ (DDBJ, 2015).

2.7 CONSIDERAÇÕES FINAIS

A complexidade existente no dogma central da biologia molecular descrito nestas sessões pressupõe aparatos tecnológicos com grande capacidade e qualidade para auxiliar os pesquisadores. O volume de dados gerado pelos seres vivos cresce de forma exponencial e requer que os bancos de dados, servidores, *softwares* e demais tecnologias sejam compatíveis com as suas necessidades de armazenamento e processamento.

Nas últimas sessões deste capítulo, fica evidente a colaboração e parceria existente entre organizações e pesquisadores no campo da biologia molecular. Suas ferramentas e conhecimentos estão acessíveis a todos. Tudo isso é tratado como um patrimônio público. Para manter tais serviços ativos requer-se uma grande estrutura, alta tecnologia, pessoal capacitado e contínua evolução do meio.

As ferramentas são diversificadas e, em alguns aspectos, únicas. As ferramentas de buscas, por exemplo, podem ser muito similares quando se fala da procura por termo como mostrado nas Figuras 11, 15 e 20, mas suas telas de resultado são distintas como visto nas Figuras 12, 16 e 22. Outas diferenças estão nas opções avançadas e nos parâmetros. Sob esse aspecto, se diferenciam pela quantidade e pelo objeto proposto pela instituição proprietária.

Nesse contexto, destaca-se o portal e a base de dados do NCBI, no qual a ferramenta de pesquisa por texto realiza a busca em todas as bases de dados disponíveis. Sua tela de resultado permite realizar filtros, minimizando a quantidade de resultados, tornando mais fácil obter a informação desejada e com maior precisão.

Tendo estes portais como modelo de funcionamento, passa-se à análise do portal IntergenicDB a fim de verificar a possibilidade de implantar melhorias em sua ferramenta de pesquisa.

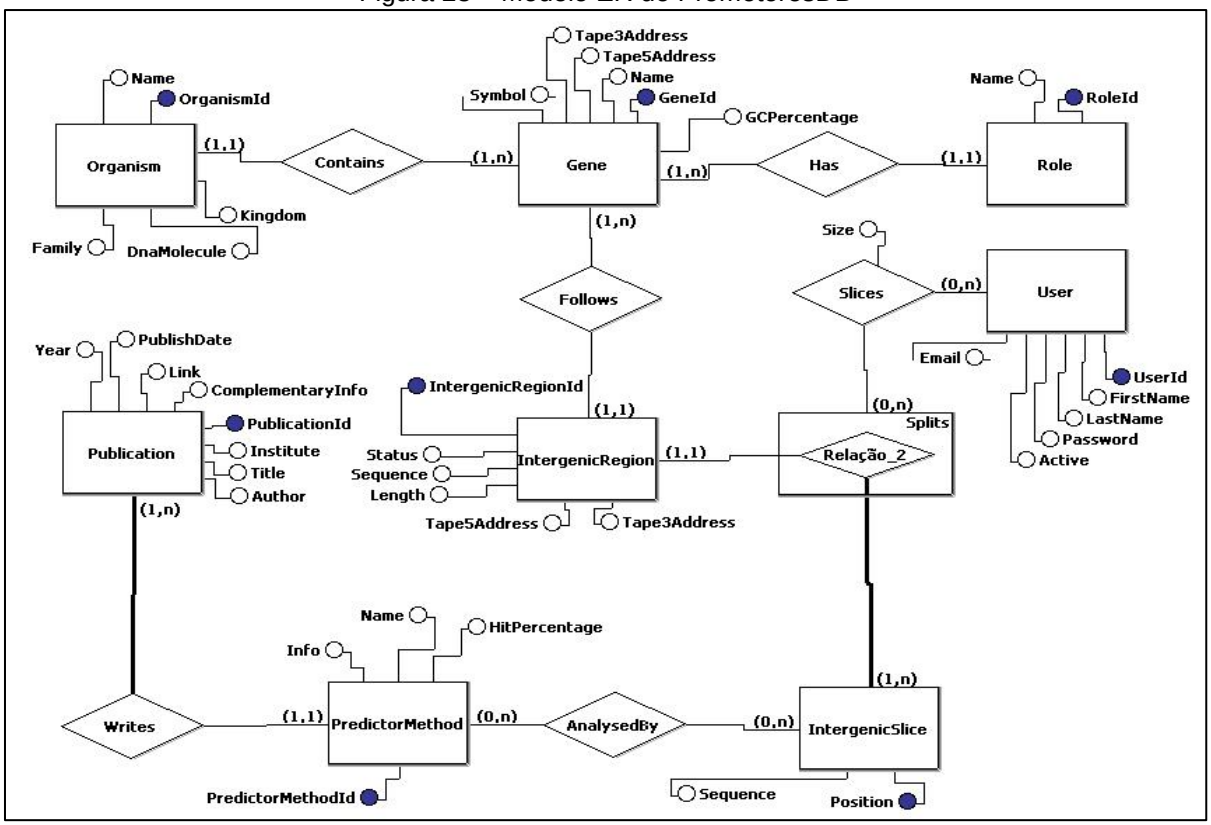
3 PORTAL INTERGENICDB

Este capítulo aborda as características do portal IntergenicDB e do banco de dados PromotoresDB, com o objetivo de fazer uma análise das ferramentas já existentes.

3.1 PROMOTORESDB

O banco de dados PromotoresDB foi criado para armazenar sequências de nucleotídeos de regiões promotoras de organismos procariontes A estrutura do PromotoresDB está representado em modelo ER, mostrado na Figura 23 (MOLIN, 2009; DALZOCHIO, 2014; INTERGENICDB, 2015).

Figura 23 – Modelo ER do PromotoresDB



Fonte: IntergenicDB[38] (2015).

[38] Disponível em: <http://intergenicdb.bioinfoucs.com/Content/images/promotoresDB_entity_relation_model.jpg>. Acessada em: 25/10/2015.

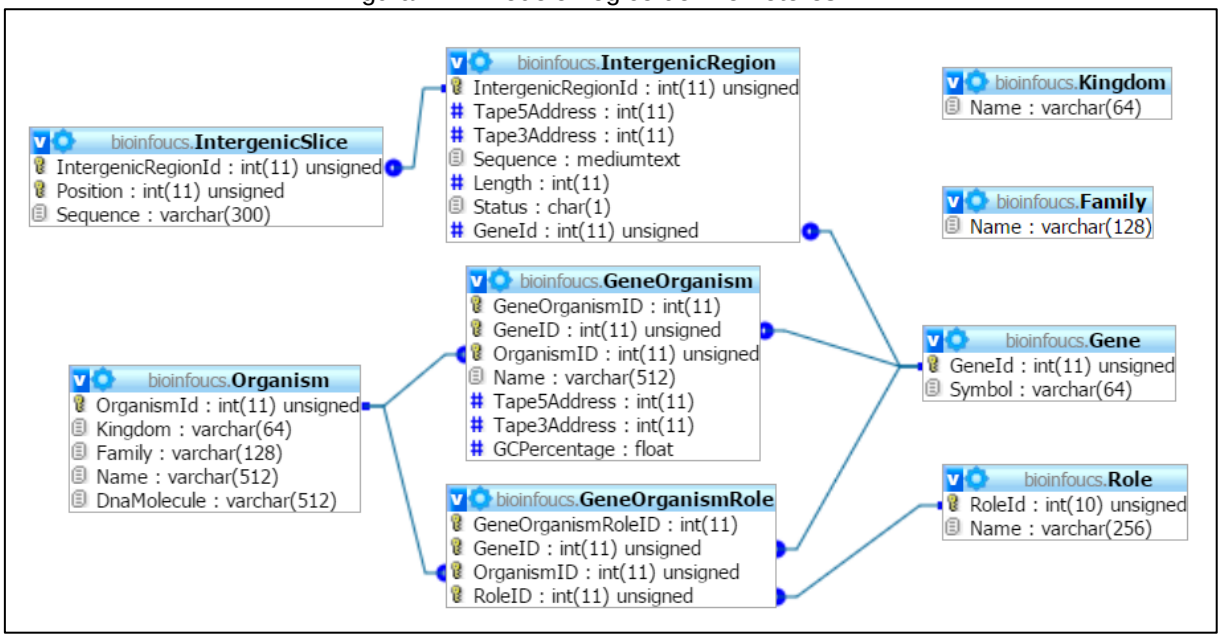
O modelo ER, Figura 23, apresenta diversas tabelas que compõem o PromotoresDB. Todas as tabelas desempenham um importante papel, mas algumas se destacam sobre as demais como, por exemplo, as tabelas Gene, Organism, IntergenicRegion e PredictorMethod. Os campos dessas tabelas são, praticamente, os mesmos usados no portal para realizar as pesquisas (DAVANZO, 2010).

A tabela *Gene* e *Organism*, são praticamente pontos de partida de uma consulta. Ambas armazenam, respectivamente, os nomes que os rotulam (no caso do Gene, os símbolos são os nomes) (DAVANZO, 2010).

Tão importante quando o cadastro (os dados), é o relacionamento das tabelas, mostrado na Figura 24, através de chave estrangeira (FK). FK é um relacionamento entre tabelas distintas que compartilham dados para manter a integridade (ELMASRI; NAVATHE, 2005; DAVANZO, 2010).

Ambas as tabelas, Gene e Organism, compartilham um de seus campos com as demais tabelas. Esses relacionamentos permitem uma melhor comunicação, simplificação na consulta em SQL, minimização no volume de dados e confiabilidade com integridade dos dados (ELMASRI; NAVATHE, 2005; DAVANZO, 2010).

Figura 24 – Modelo Lógico do PromotoresDB



Fonte: Da Rosa (2015).

A relação, Figura 23, entre três das quatro principais tabelas é direta, ligadas através de um campo de identificação. O relacionamento da tabela *PredictorMethod* possui um caminho maior, necessitando acessar informações da tabela

IntergenicSlice combinada com a tabela *IntergenicRegion*, o que pode tornar a consulta mais lenta, na hipótese do usuário utilizar os parâmetros que correspondem à essa tabela (DAVANZO, 2010; PICOLOTTO, 2012; DALZOCHIO, 2014).

O relacionamento das tabelas Organism, Intergenicregion e Role com a tabela Gene é de um para muitos (1,n). Visualmente, pressupõe-se que a tabela Gene possui maior número de linhas do banco de dados, entretanto, a maior delas é a Intergenicregion[39] que, por sua vez, possui a principal informação da base de dados (DAVANZO, 2010; PICOLOTTO, 2012; DALZOCHIO, 2014).

Outra observação sobre o modelo lógico, Figura 24, é sobre dois objetos nomeados de *Family* e *Kingdom*. Ambos os objetos não são tabelas, são *views*.

Uma *view*, ou visão, é uma tabela virtual, ou seja, é um alias para uma consulta SQL. Toda vez que a view é executada, o comando SQL é executado. A *query* SQL que gera a *view* é chamada de *SimpleSearch*, conforme se verifica na Figura 25 (ELMASRI; NAVATHE, 2005).

Figura 25 – Query SQL da *view* SimpleSearch

```
CREATE ALGORITHM = UNDEFINED DEFINER = `bioinfoucs`@`%` SQL SECURITY INVOKER VIEW `SimpleSearch` AS
SELECT `o`.`OrganismId` AS `OrganismId`, `g`.`GeneId` AS `GeneId`,
    `i`.`IntergenicRegionId` AS `IntergenicRegionId`, `r`.`RoleId` AS `GeneRoleId`,
    `o`.`Name` AS `OrganismName`, `o`.`Kingdom` AS `OrganismKingdom`,
    `o`.`Family` AS `OrganismFamily`, `o`.`DnaMolecule` AS `OrganismDnaMolecule`,
    `go`.`Name` AS `GeneName`, `g`.`Symbol` AS `GeneSymbol`,
    `r`.`Name` AS `GeneMainRole`, `go`.`Tape5Address` AS `GeneTape5Address`,
    `go`.`Tape3Address` AS `GeneTape3Address`, `go`.`GCPercentage` AS `GeneGcPercentage`,
    `i`.`Length` AS `IntergenicRegionSize`, `i`.`Status` AS `IntergenicRegionTapeDirection`,
    `i`.`Tape5Address` AS `IntergenicRegionTape5Address`,
    `i`.`Tape3Address` AS `IntergenicRegionTape3Address`,
    `i`.`Sequence` AS `IntergenicRegionSequence`, `pm`.`Name` AS `PredictorMethodName`,
    `pm`.`HitPercentage` AS `PredictorMethodHitPercentage`,
    `is`.`Position` AS `IntergenicSlicePosition`, `is`.`Sequence` AS `IntergenicSliceSequence`
FROM
    (((((((`Organism` `o`
    JOIN `GeneOrganismRole` `gor` ON ((`gor`.`OrganismID` = `o`.`OrganismId`)))
    JOIN `GeneOrganism` `go` ON (((`gor`.`OrganismID` = `go`.`OrganismID`)
        AND (`gor`.`GeneID` = `go`.`GeneID`))))
    JOIN `Gene` `g` ON ((`g`.`GeneId` = `gor`.`GeneID`)))
    JOIN `Role` `r` ON ((`gor`.`RoleID` = `r`.`RoleId`)))
    JOIN `IntergenicRegion` `i` ON ((`i`.`GeneId` = `g`.`GeneId`)))
    LEFT JOIN `IntergenicSlice` `is` ON ((`i`.`IntergenicRegionId` = `is`.`IntergenicRegionId`)))
    LEFT JOIN `PredictorMethod_IntergenicSlice` `p`
        ON (((`p`.`IntergenicRegionId` = `is`.`IntergenicRegionId`)
        AND (`p`.`Position` = `is`.`Position`))))
    LEFT JOIN `PredictorMethod` `pm` ON ((`pm`.`PredictorMethodId` = `p`.`PredictorMethodId`)));
```

Fonte: Da Rosa (2015).

[39] Disponível em: <http://mysql04-farm13.kinghost.net/db_structure.php?db=bioinfoucs>. Acessada em: 25/11/2015.

Para a implementação do PromotoresDB foi escolhido o SGBD MySQL. Os motivos para a escolha dessa ferramenta são variados, podendo destacar a sua gratuidade e a popularidade. Aspectos esses importantes, pois influenciam no custo de um projeto e no suporte (MOLIN, 2009).

3.2 INTERGENICDB

O portal IntergenicDB (Figura 26) é um repositório de acesso público no qual pesquisadores podem ter acesso às informações do banco de dados PromotoresDB, desenvolvido para ser uma ferramenta de consulta ao banco de dados e para o *download* e *upload* de arquivos (PICOLOTTO, 2012; DALZOCHIO, 2014; NOTARI et al., 2014).

Figura 26 – Página inicial do portal IntergenicDB

Fonte: IntergenicDB[40] (2015).

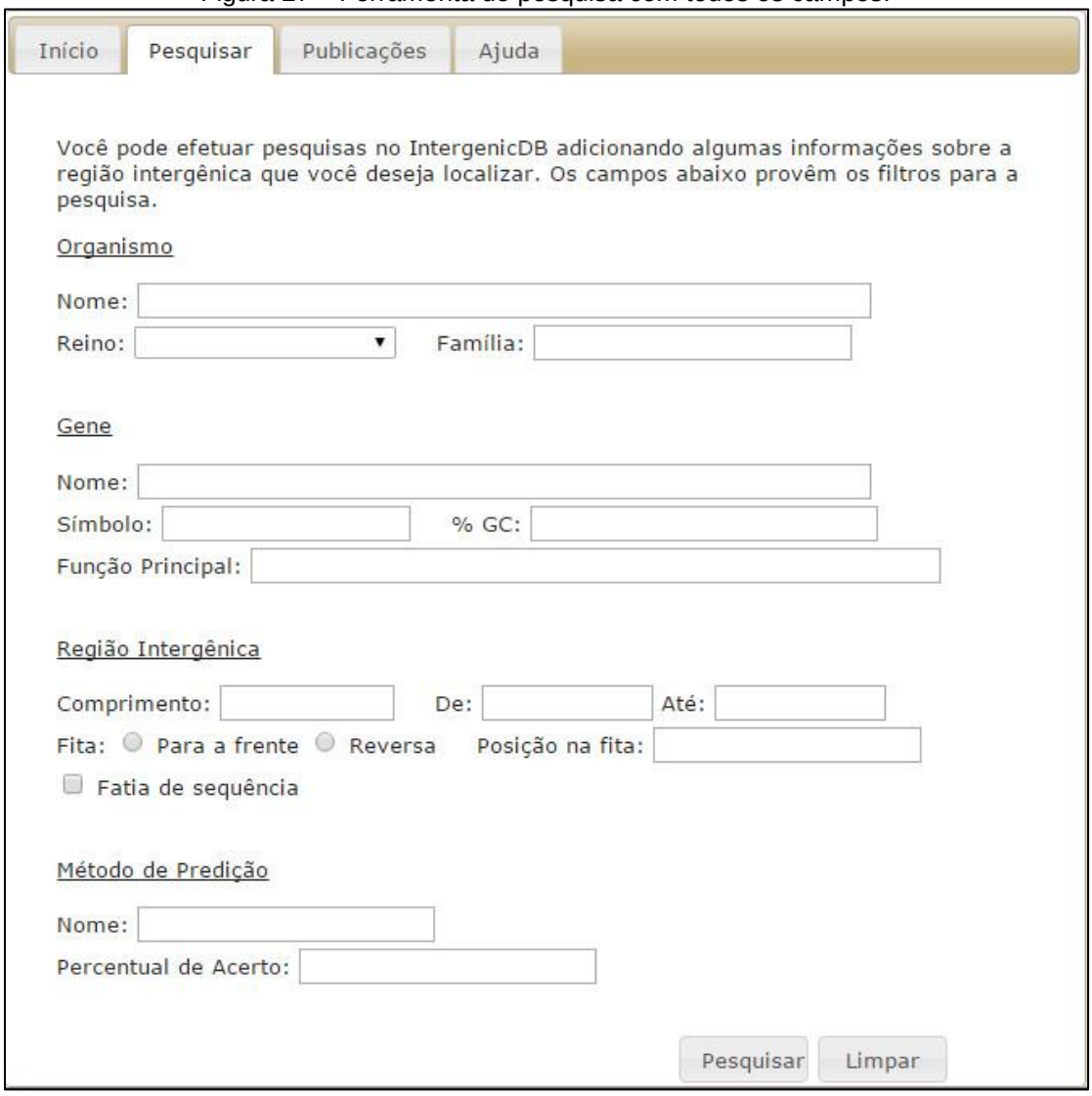
O acesso ao portal é público, permitindo que qualquer usuário utilize livremente os recursos de consulta. O resultado da consulta para um usuário não registrado

[40] Disponível em: <http://intergenicdb.bioinfoucs.com>. Acessada em: 25/10/2015.

somente é disponibilizado em tela. Para o *download* da informação é necessário possuir usuário e senha (DAVANZO, 2010; PICOLOTTO, 2012; DALZOCHIO, 2014).

A ferramenta de busca do site propõe, inicialmente, que a pesquisa seja realizada pelos campos que compõem o grupo Organismos. Para ter acesso aos demais parâmetros de pesquisa, é necessário clicar sobre o nome dos demais grupos: Gene, Região Intergênica e Método de Predição. Com isso, há possiblidade de realizar uma pesquisa com outros parâmetros, independentes ou em conjunto. Todos os parâmetros podem ser vistos na Figura 27 (DAVANZO, 2010).

Figura 27 – Ferramenta de pesquisa com todos os campos.



Fonte: IntergenicDB[41] (2015).

[41] Disponível em: <http://intergenicdb.bioinfoucs.com/Default/Search>. Acessada em: 31/10/2015.

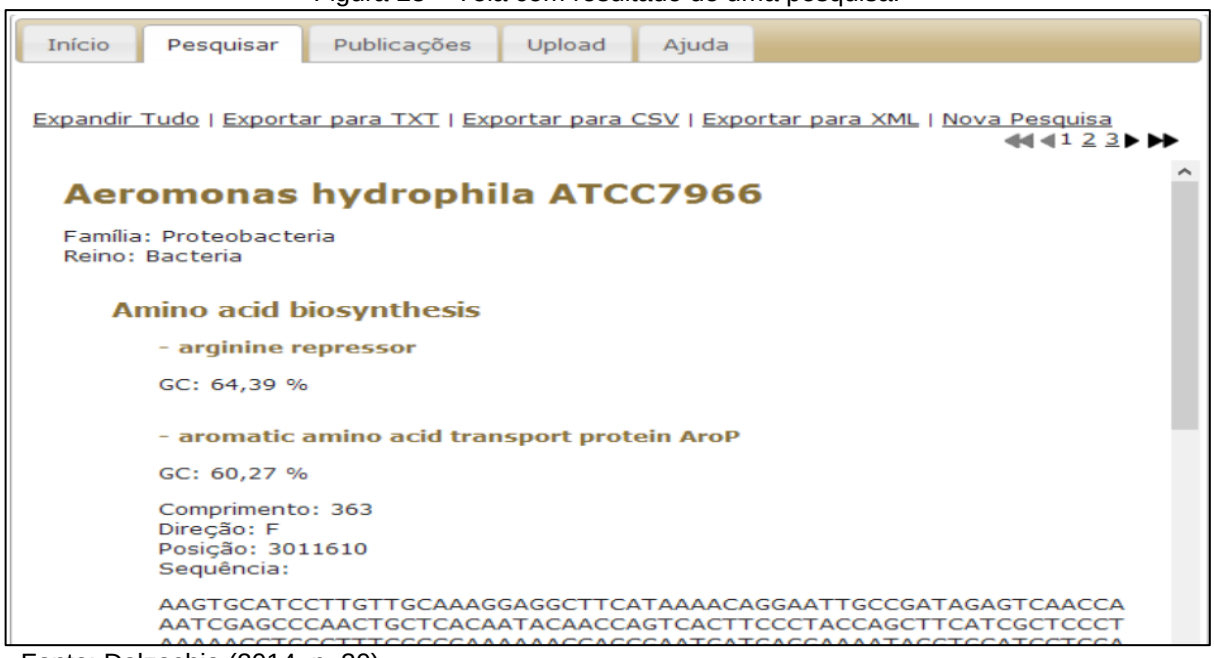
Para realizar a pesquisa: basta preencher um ou mais campos e clicar no botão “Pesquisar”. Alguns campos como, por exemplo, “Nome” do grupo Organismo possuem o recurso de auto completar.

Em comparação com as ferramentas de pesquisa do NBCI, ENA e DDBJ, a ferramenta do portal IntergenicDB não possui recursos que permitam ampliar – mais de um parâmetro por campo – e filtrar uma pesquisa simultaneamente. Pode-se citar, como exemplo, a ausência dos seguintes recursos:

a) pesquisar pelo valor X e Z, simultaneamente, no mesmo campo. Esse tipo de pesquisa seria uma união de termos (*and*), onde é necessário a existência dos dois valores;
b) pesquisar pelo valor X e Z, simultaneamente, no mesmo campo. Esse tipo de pesquisa seria uma disjunção (*or*), bastando existir somente um dos valores;
c) pesquisar por qualquer valor, exceto Y. Recurso de negação (*not*) para excluir do resultado valores indesejáveis no campo em questão;

O resultado em tela é paginado, ou seja, somente será visível um organismo por vez. Conforme a Figura 28, a consulta realizada retornou organismos e está sendo visualizado o primeiro. A paginação é organizada alfabeticamente pelo nome dos organismos presentes no resultado da pesquisa.

Figura 28 – Tela com resultado de uma pesquisa.

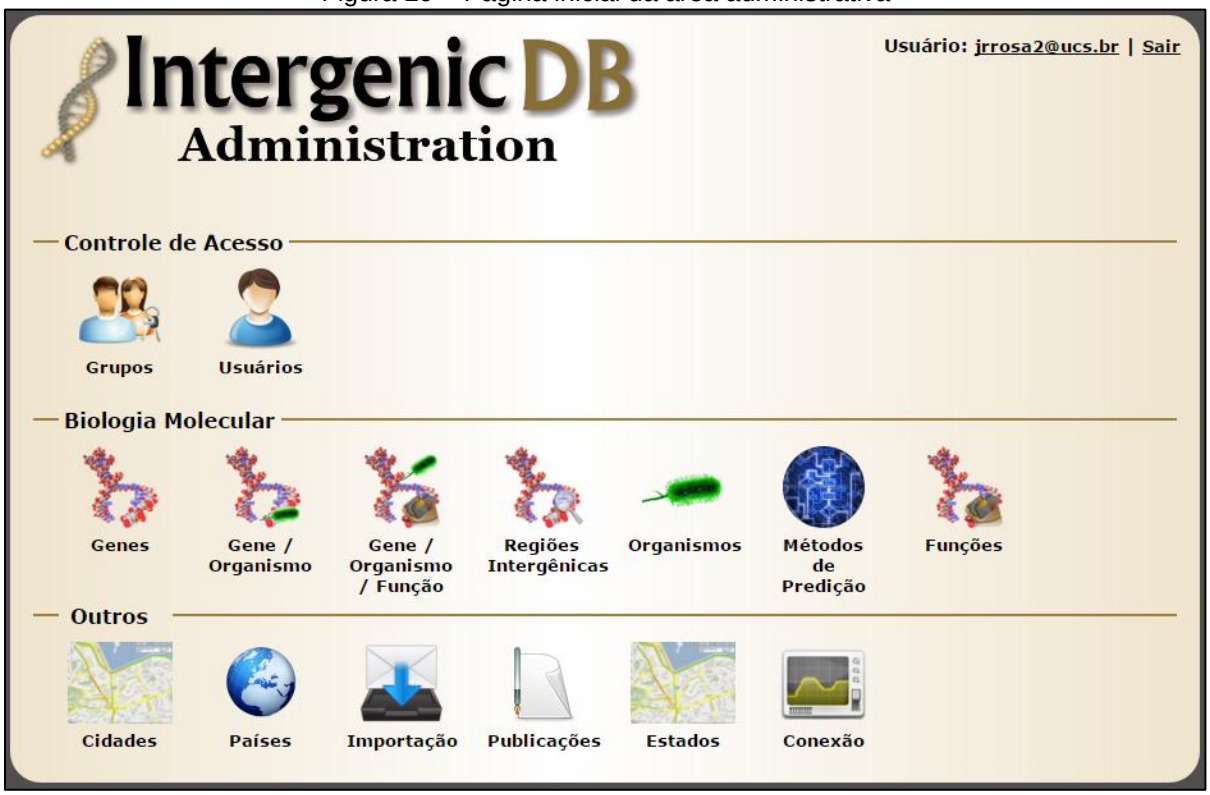


Fonte: Dalzochio (2014, p. 30).

Esse tipo de visualização, apesar de limpa e objetiva, limita a possibilidade de comparação entre os resultados, pois não é possível visualizar vários organismos simultaneamente. Isso poderia ser melhorado caso os dados fossem dispostos numa tabela.Para tanto, é necessário que o utilizador faça o *download* do resultado a partir dos *links* disponíveis na tela. São três opções de download e cada uma dispõe de formatos de arquivos textos diferentes: TXT, CSV e XML.O arquivo conterá todas as informações resultantes da pesquisa, incluindo os dados paginados que não estão visíveis.

Outra parte do portal é a área administrativa (Figura 29), desenvolvida pelo aluno Douglas Picolotto (PICOLOTTO, 2012). O acesso é restrito e requer que o novo utilizador realize um cadastro e que seja devidamente autorizado por um dos membros com tal privilégio.

Figura 29 – Página inicial da área administrativa



Fonte: IntergenicDB[42] (2015).

Na Figura 29, pode-se ver que os diversos itens estão separados em três grupos: Controle de Acesso, Biologia Molecular e outros. O primeiro grupo está ligado

[42] Disponível em: <http://intergenicdb.bioinfoucs.com/admin>. Acessado em: 05/11/2015.

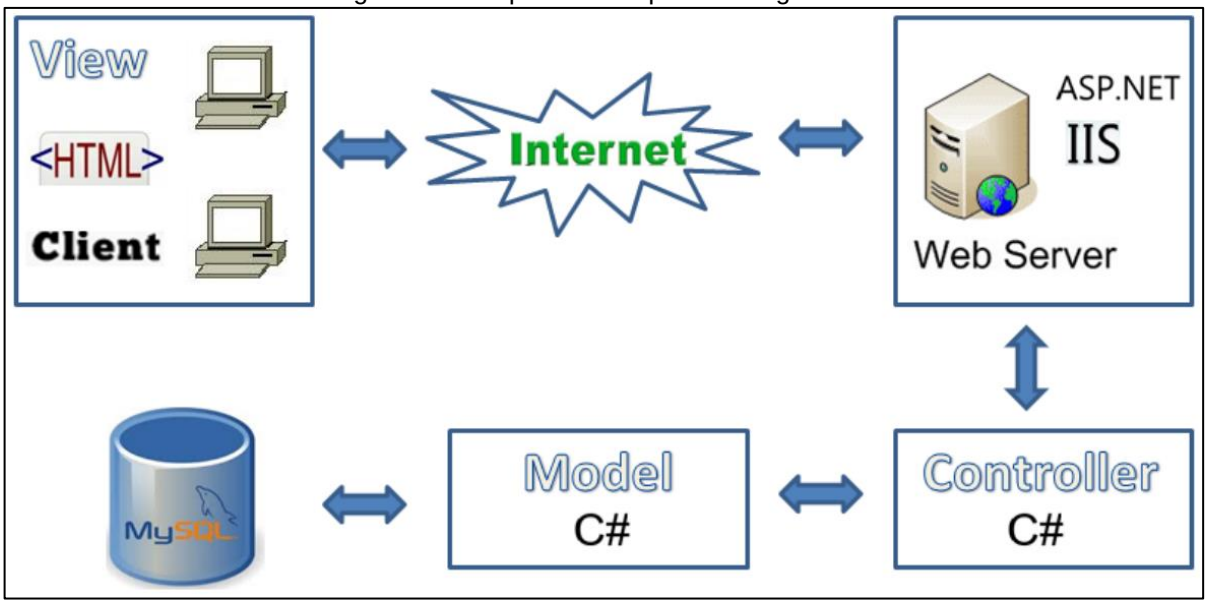
ao cadastro e gestão de usuário e grupos. Um usuário pertence a um grupo que lhe proverá mais ou menos acessos. Na sequência da Figura 29, os itens do grupo Biologia Molecular estão diretamente ligados à gestão dos dados biológicos inseridos no banco de dados. Através deles é possível inserir, editar ou excluir informações. Apesar da ferramenta permitir a inserção, seu principal objetivo é a manutenção (PICOLOTTO, 2012).

O grupo Outros, é mais heterogêneo. Os itens Cidade, Países, Estados e Conexão estão ligados à geoposição do usuário que utilizar o portal. Essas informações são utilizadas somente para cálculos e estatísticas.

O item Importação é a ferramenta que, apesar do nome, não inclui dados externos no banco de dados. Ela é utilizada pelos administradores para visualizar as novas informações que foram importadas para o PromotoresDB. Entretanto, somente após a validação do administrador, essa informação nova será visível na tela de consulta. Caso contrário, ela pode ser excluída do banco.

O portal está armazenado em um provedor que comporta o SGBD MySQL e o Web Server IIS. Para a criação do site foi utilizada a plataforma de desenvolvimento web da Microsoft, a ASP .NET usando a linguagem de programação *C Sharp* (C#) e a padrão de arquitetura MVC. A arquitetura está representada na Figura 30 (DAVANZO, 2010; DALZOCHIO, 2014).

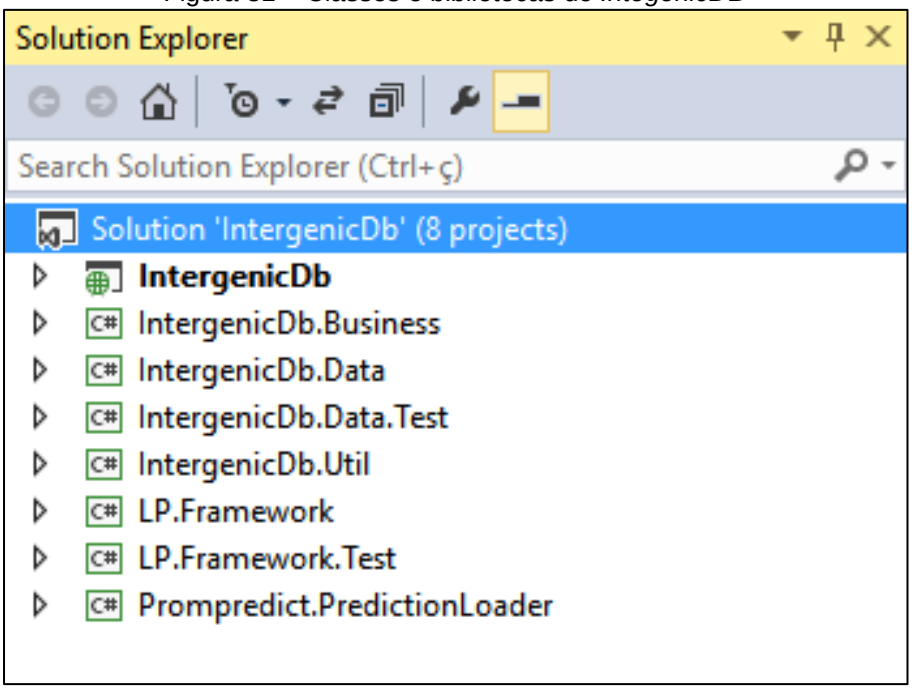
Figura 30 – Arquitetura do portal IntergenicDB



Fonte: Dalzochio (2010, p. 27).

O código fonte do portal é formado por bibliotecas (Figura 31) que, por sua vez, são um conjunto de classes. A Figura 31 trata-se de uma sub tela da *IDE* de desenvolvimento da Microsoft, o Visual Studio 2013. Cada grupo desempenha uma ou mais funções para atender os padrões de arquitetura e, por fim, constituir o portal (DAVANZO, 2010).

Figura 31 – Classes e bibliotecas do *IntegenicDB*



Fonte: Da Rosa (2015).

A biblioteca IntergenicDB foi desenvolvida usando o padrão MVC, mas somente contém as *Views* e as *Controllers*. As demais bibliotecas formam a *Model.* A classe mais importante do IntergenicDB é a *DefaultController*, que possui as *Actions*. Ou seja, para cada ação do usuário na *View*, haverá diferentes métodos (*Actions*) que retornarão para o usuário uma *View.* A *SearchResult* é a *Action* que contém a lógica que realiza as buscas no banco de dados e retorna os resultados. Essa ação ocorre através da utilização das bibliotecas que estão implementas como *Model* (DAVANZO, 2010).

Na biblioteca IntergenicDB.Business foram definidos os objetos. Cada classe (objetos) dessa biblioteca representa as tabelas e *views* do banco de dados. A classe possui a declaração de cada campo, o tipo de dado e atributos que definem se o campo é somente leitura ou também é escrita (DAVANZO, 2010).

A *LP.Framework* é um conjunto de classes que, além de atuarem como *Model* do padrão MVC, também recebeu os padrões *Abstract Factory* e *Data Gateway*. Esses padrões podem ser encontrados nas classes:

a) *IDataGateway*: é uma interface utilizada pelas *Controllers*, que define como os objetos acessam os dados. Utiliza o padrão *Data Gateway*;
b) *DataGatewayFactory*: é uma classe responsável por criar as instâncias de conexão, permitindo que as *Controllers* utilizem qualquer *Data Gateway*, desenvolvida usando o padrão *Abstract Factory*;
c) *IDataModel*: é uma interface para criação dos objetos na IntergenicDB.Business. Ela segue o padrão Data Gateway (GAMMA, 2000; FOWLER et al., 2006; DAVANZO, 2010).

Outras classes importantes do *LP.Framework* são a *Predicate* e a *SearchPredicateBuilder*. Ambas são responsáveis por construir as *querys* (consultas) em SQL que serão executadas no SGBD. Através de chamadas de métodos, elas recebem parâmetros dos *Data Gateway* e traduzem para a linguagem SQL, gerando a *string* de execução (FOWLER et al., 2006; DAVANZO, 2010).

3.3 CONSIDERAÇÕES FINAIS

Tanto o portal IntergenicDB e o banco de dados PromotoresDB são ferramentas que surgiram para atender a necessidade do grupo de Bioinformática da Universidade de Caxias do Sul (UCS). O desenvolvimento foi realizado por alunos em seus trabalhos de conclusão de curso como, por exemplo, Aurione Molin (MOLIN, 2009), Vanessa Davanzo (DAVANZO, 2010), Douglas Picolotto (PICOLOTTO, 2012) e Jovani Dalzochio (DALZOCHIO, 2014). Todos os colegas contribuíram significativamente para a excelência da ferramenta.

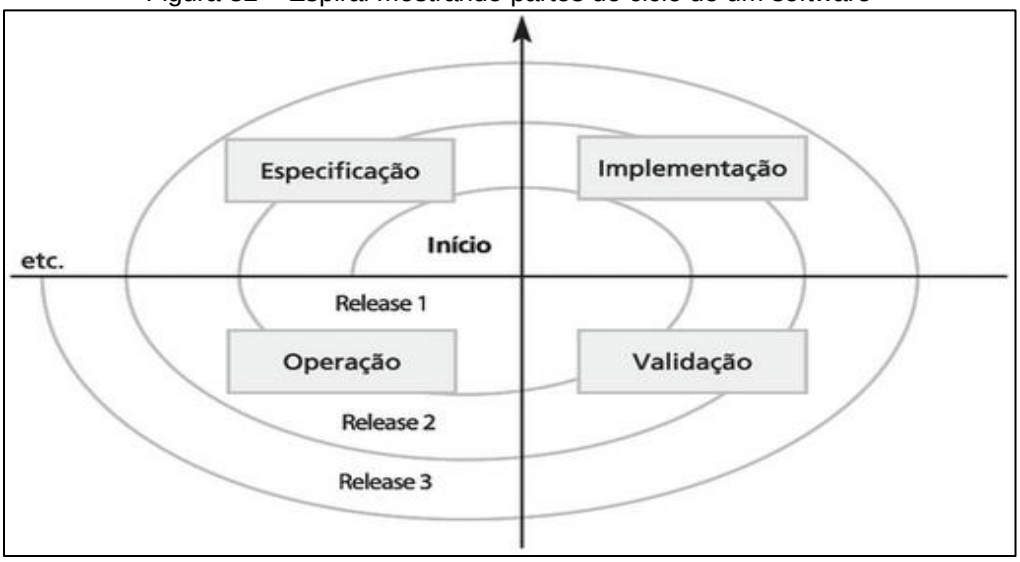
Atualmente, devido ao crescimento da base de dados, foram detectadas lentidões, timeouts e inconsistência nos resultados que interferem na utilização do portal. A partir desses problemas foram levantados requisitos de melhoria que tornaram a ferramenta mais eficaz e estável. Esses requisitos são apresentados no capítulo 4, Proposta de solução.

4. PROPOSTA DE SOLUÇÃO

Atualmente, o portal tem apresentado problemas de lentidão para a retornar uma pesquisa e a usabilidade da ferramenta, tanto para executar a pesquisa como manuseá-la, não é amigável. Reparos são necessários para a melhoria do portal, mas seu código fonte e bibliotecas são complexos.

Todo o sistema necessita de reparos para erros que são encontrados; de adaptação quando há mudança de ambiente; ou, melhorias quando a regra de negócio muda gerando novos requisitos. Durante esse tempo, o processo de criação/melhoria pode ser ilustrado numa espiral, conforme a Figura 32 (SOMMERVILLE, 2011).

Figura 32 – Espiral mostrando partes do ciclo de um software



Fonte: Sommerville (2011, p. 165).

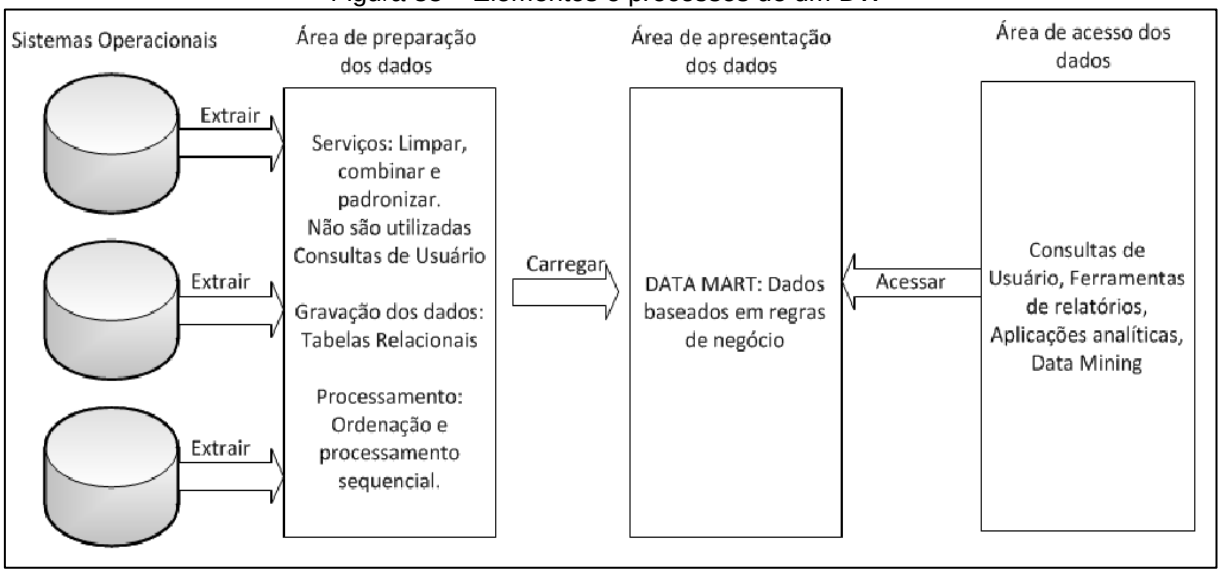
O ciclo inicia com o levantamento de requisitos (as necessidades), a elaboração do projeto (proposta), a implementação (desenvolver o que foi proposto), os testes (verificar se atende aos requisitos) e, finalmente, entregar o produto para o usuário utilizar. Assim como na maioria dos softwares, o portal IntergenicDB precisa atender alguns requisitos para se tornar uma ferramenta eficiente e intuitiva, tais como: velocidade, confiabilidade, facilidade de uso e tamanho (SOMMERVILLE, 2011).

Dentre as melhorias aplicáveis a um software em evolução, a implementação de um Data Warehouse (DW) que é um processo que, a partir de dados externos, extrairá, limpará e transformará as informações para uma nova base de dados

dimensionada e consolidada, modelando e reunindo os dados a fim de obter uma melhor performance (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

A Figura 33, descreve os elementos e os processos básicos de um DW, exemplificando uma origem dos dados, os processos de limpeza e transformação e finalizando com a carga dos dados na nova base. A partir desse ponto, os acessos aos dados serão realizados através da nova base (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

Figura 33 – Elementos e processos de um DW



Fonte: Adaptado de Kimball; Ross (2002, p. 9).

Diferente do modelo entidade e relacionamento (ER), utilizado atualmente pelo PromotoresDB (Figura 23), para desenvolver a solução, será abordada uma modelagem comumente utilizada em DW: a modelagem dimensional estrela (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

O modelo dimensional estrela é uma técnica que remove normalizações da origem (fonte dos dados) ao importar os dados para a uma nova tabela que será criada. O nome modelo estrela deve-se à forma visual como o esquema é apresentado: uma tabela central envolta a outras tabelas, das quais serão minerados os dados (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

Mencionado anteriormente, a que será criada e a tabela central, ambas são o mesmo objeto do modelo estrela, conhecida como **tabela fato**. A tabela fato, é a que receberá os dados importados. Segundo Kimball (2002, p. 21): “[...] é a principal tabela

de um modelo dimensional em que as medições numéricas de desempenho da empresa são armazenadas [...]" (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

Dentro do conceito de modelo estrela, há tabelas que circundam a tabela fato são chamadas de **tabelas dimensão**, fontes de origem dos dados. Lecionado por Kimbal (2002, p. 25), "As tabelas de dimensão são pontos de entrada para a tabela de fatos. Atributos de dimensões eficazes produzem recursos [...] eficazes." (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

Geralmente, as tabelas dimensões são constituídas de diversas colunas e atribuídos, do quais nem todos são necessários para o resultado de uma consulta. Por isso, os dados mais importantes serão centralizados em um único local, minimizando a complexidade da *query*, diminuindo os inúmeros acessos a outras tabelas e simplificando as modificações que serão necessárias realizar no portal IntergenicDB (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

O DW possui metodologia própria, sendo que sua modelagem dimensional visa priorizar o desempenho nas consultas. Para isso, a técnica a ser aplicada deve respeitar:

a) a criação de modelos simples, extraindo somente os dados necessários;
b) os dados devem estar consistentes, um processo de limpeza se encarrega de eliminar dados incorretos que possam prejudicar o objetivo;
c) a importação dos dados para nova base precisa ser a mais direta possível, evitando conversões e outras técnicas que possam causar perda na precisão dos dados;
d) quaisquer alterações nos dados devem, preferencialmente, ser realizadas na origem, não no destino (KIMBALL; ROSS, 2002; GOIS; 2004; SATOR, 2004; VIEIRA, 2012).

Com base na observação desses aspectos, as próximas sessões serão dedicadas a apresentar os novos requisitos e as modificações que serão realizadas para atender às novas necessidades de pesquisa no portal.

4.1 REQUISITOS

Nessa sessão os requisitos são organizados em subseções, aonde títulos indicam qual requisito será descrito. Dentro de cada requisito foi estruturado para uma melhor organização dos conceitos, dividindo-os em:

a) Problema – descrição do problema e consequências;
b) Proposta – apresentação da proposta para solucionar o problema descrito;
c) Testes – métodos e técnicas para avalizar se a solução atende os requisitos conforme apresentado.

4.1.1 Ajustes na base de dados para a consulta

4.1.1.1 Problema

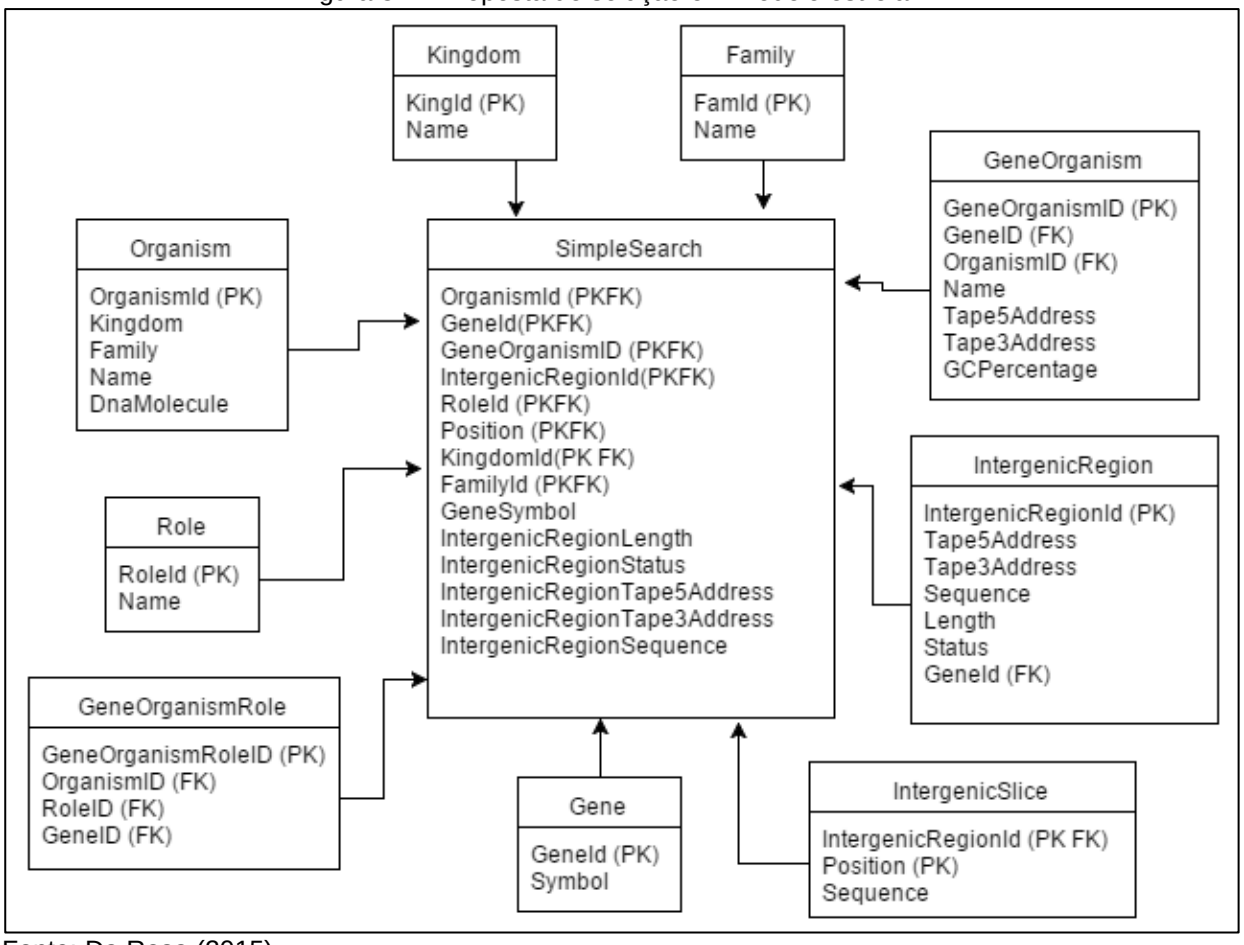
Apesar do projeto atual do PromotoresDB estar devidamente estruturado, o *LP.Framework* não utiliza todos os recursos disponíveis para realizar uma consulta. Todas as informações solicitadas na tela de consulta são extraídas de uma *view*, a *SimpleSearch* (DAVANZO, 2010; DALZOCHIO, 2014).

De acordo com a Figura 25, verifica-se que a *query* acessa nove tabelas diferentes e considerando o funcionamento de uma *view*, para cada requisição feita, à tabela *SimpleSearch*, o tamanho da base de dados irá dobrar no momento do acesso. Como consequência, o sistema apresenta lentidão e *timeout*, podendo em algum momento, ultrapassar o limite do espaço em disco disponível. Outro fator é a complexidade da *query*, que exigirá consumir muitos dos demais recursos do provedor onde está hospedado (ELMASRI; NAVATHE, 2005; DAVANZO, 2010; DALZOCHIO, 2014).

4.1.1.2 Proposta

Para sanar o problema propõe-se a criação de uma nova tabela segundo os conceitos de *Data Warehouse* (DW) e o modelo dimensional estrela. Uma possível solução a essa proposta está ilustrada na Figura 34.

Figura 34 – Proposta de solução em modelo estrela.



Fonte: Da Rosa (2015).

Dentro da base de dados atual, a PromotoresDB, será criada a tabela fato conforme a Figura 34, que demonstra o esquema de solução proposto para o caso. A criação da tabela, extração dos dados, limpeza e importação será através de consultas em SQL executadas no banco de dados. Outra alteração será a transformação das *views* Kingdom e Family para tabelas.

Todas as consultas para a implementação serão desenvolvidas e testadas em um ambiente controlado, numa base de dados de testes. Nenhuma alteração será executada na base de dados online até a validação dos testes serem concluída.

Não será criado um novo banco de dados para o Data Warehouse. A tabela fato coexistirá na mesma base de dados e as tabelas da estrutura serão utilizadas como tabelas dimensões.

4.1.1.3 Testes

Para validar o novo modelo proposto, serão executados os seguintes testes:

a) criar script SQL para gerar a tabela fato;
b) testar o desempenho do script;
c) criar uma view, similar a antiga SimpleSeach, usando somente as tabelas dimensões;
d) simular consultas, na *view* e o *datawarehouse,* para medir o tempo de execução e comparar o desempenho.

Como resultado dos testes, deve-se obter as mesmas linhas de dados ao executar as consultas na tabela fato e nas tabelas dimensões. Outro aspecto esperado é que o tempo de execução da tabela fato seja inferior, ou seja, mais rápido, que ao ser executado nas tabelas dimensões.

4.1.2 Alteração do mecanismo de acesso ao banco de dados

4.1.2.1 Problema

O mecanismo atual, o *LP.Framework*, utiliza padrões de arquitetura de software que o tornaram complexo. Mudanças significativas na sua lógica podem afetar tanto o mecanismo de consulta quanto outras páginas do portal que utilizam essa biblioteca como, por exemplo, a área administrativa.

Outro problema está na inconsistência dos dados na tela de resultado. Em alguns casos, os parâmetros informados para pesquisa não correspondem ao

resultado em tela. Isso sugere erro de lógica nas classes ou nas interfaces das bibliotecas que compõem a *Model* do projeto.

4.1.2.2 Proposta

Para atender a esse requisito, serão propostas duas alternativas:

a) realizar alterações nas bibliotecas que compõem o mecanismo de busca;
b) implementar a biblioteca *Entity Framework* (EF), para criar um novo mecanismo de busca.

Inicialmente, será verificado se o *LP.Framework* efetuará acessos à nova tabela fato, utilizando o mesmo nome e campos da *view SimpleSearch*. Essa validação servirá como um indicador das possíveis mudanças que serão necessárias realizar nesse *framework*. Serão necessários e aceitos pequenos ajustes no código fonte que não causem impacto ao projeto.

Num segundo momento, avaliar os erros e definir as mudanças que devem ser feitas no projeto atual. O resultado da avalição deverá focar-se no principal item de avaliação: as mudanças não poderão afetar o acesso ao banco das demais páginas do portal.

No próximo passo, será verificada a compatibilidade do *Entity Framework* com o projeto atual.

O *Entity Framework* é uma biblioteca de manipulação de dados através de classes e objetos, muito similar à *LP.Framework* pois também trabalha com o padrão MVC. Sua vantagem é ser mantida e desenvolvida pela Microsoft, o que a torna fácil de integrar com as plataformas .NET, das quais foram utilizadas desenvolver o portal IntergenicDB. Outra vantagem é por ser uma biblioteca mais popular, permitindo ter maior acesso a conteúdo didáticos e suporte técnico (SANCHES; ALTHMANN, 2013).

Continuando o processo, será avaliado o impacto das mudanças ao utilizar o *Entity Framework*, usar os mesmos critérios avaliativos utilizados no LP.Framework.

O último passo é definir qual biblioteca será o novo mecanismo de consulta.

4.1.2.3 Testes

Para validar qual mecanismo será adotado, primeiramente, ele não deve prejudicar as demais *Controllers* que utilizam o mecanismo atual.

Segundo quesito de avaliação será quanto a quantidade e complexidade das mudanças que devem ser realizadas, uma vez que ambas as alternativas atendam o primeiro requisito.

4.1.3 Reconstrução da tela para o novo mecanismo de consulta

4.1.3.1 Problema

A atual interface de pesquisa, em comparação com as ferramentas do NBCI, ENA e DDBJ, não permite que o utilizador realize pesquisas múltiplas numa única consulta. A minimização do resultado, preenchendo outros campos e criando um filtro, também é limitada pelo mesmo motivo.

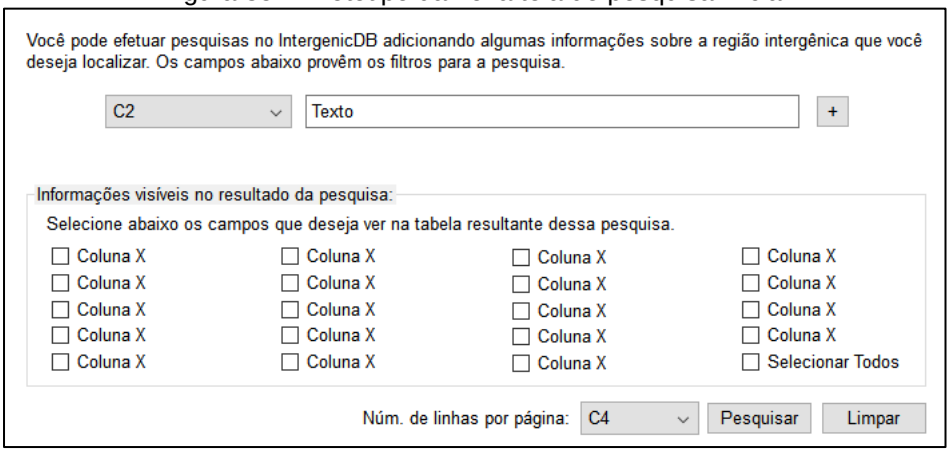
As modificações propostas na base de dados, criando um DW, requerem uma remodelagem para adequar a interface de pesquisa às modificações que serão realizadas. Entretanto essa melhoria interna na base de dados implica na necessidade uma interface que corresponda às possibilidades de exibição de resultados que serão geradas, as quais seriam inviáveis para o modelo atual

4.1.3.2 Proposta

A nova interface de pesquisa está adequada para o uso do DW e segue os padrões dos portais do NBCI, ENA, DDBJ.A tela de pesquisa manterá o *layout* atual – os padrões de CSS não serão alterados – e todos os campos de pesquisa serão

mantidos, mas não ficarão visíveis. Ao acessar a tela, somente um campo estará disponível, conforme o protótipo apresentado na Figura 35. Para cada campo adicional a ser incluído na pesquisa, o utilizador deverá clicar com o mouse no botão [+].

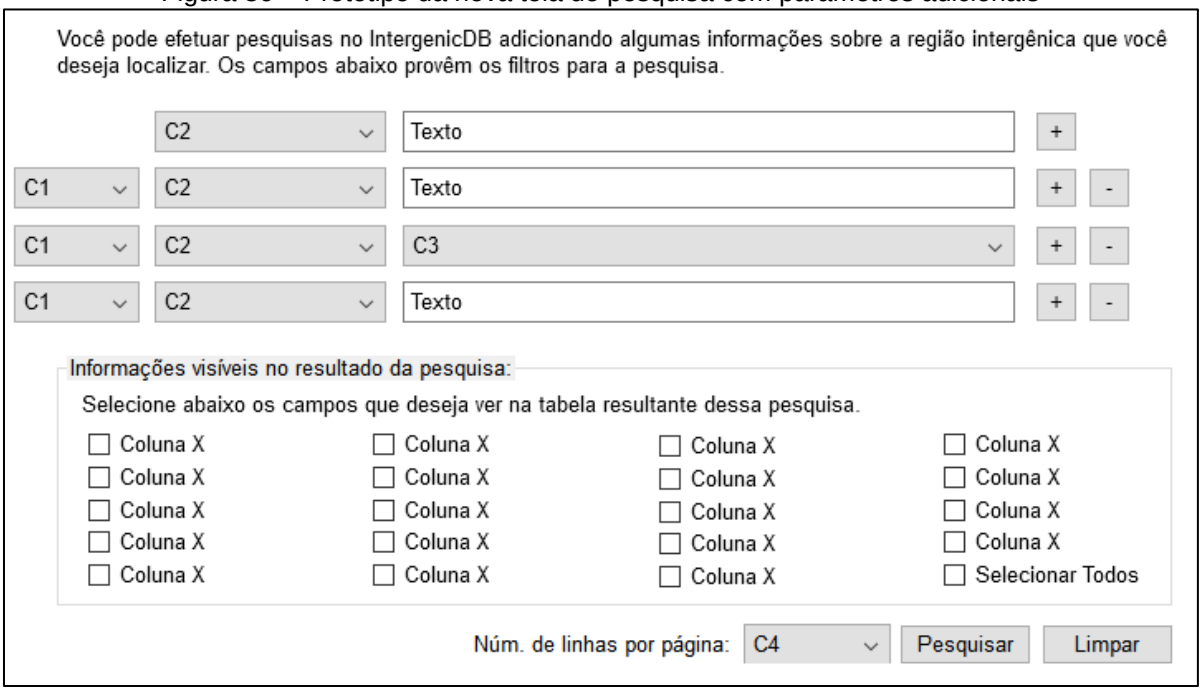
Figura 35 – Protótipo da nova tela de pesquisa inicial



Fonte: Da Rosa (2015).

Para cada campo que o utilizador deseja remover da pesquisa, será disponibilizado o botão [-]. Somente o primeiro campo não pode ser excluído. A tela de pesquisa com mais parâmetros está apresentada na Figura 36.

Figura 36 – Protótipo da nova tela de pesquisa com parâmetros adicionais



Fonte: Da Rosa (2015).

O combo box, campo nominado de “C2”, conterá uma lista com todos os campos disponíveis para parametrizar a pesquisa. O campo “Texto” será para a digitação do termo a ser pesquisado no parâmetro selecionado em “C2”. O campo “C3”, substituirá o campo “Texto” quando o parâmetro de pesquisa em “C2” possuir textos pré-definidos.

O campo “C1” exibirá três opções: *AND*, *OR* e *NOT*, permitindo que os termos pertençam, ou não, ao resultado da pesquisa. Assim, um filtro com tais opções irá garantir resultados mais precisos.

Na parte inferior da tela, serão disponibilizados vários *check box* nomeados de “Coluna X”. O utilizador irá selecionar os campos para informar ao programa quais colunas deseja visualizar na tela de resultado. Isso permitirá que o volume de dados seja menor, tornando a pesquisa mais rápida e o resultado mais objetivo para o usuário.

O *combo box* C4, por sua vez, permitirá o utilizar selecionar a quantidade de linhas que serão exibidas por página. Possuirá um valor padrão estático e demais quantidades serão múltiplos do valor padrão.

O botão nomeado de “Limpar” fará a tela retornar ao estado exibido na Figura 36 e 35. O botão “Pesquisar” irá executar a consulta conforme os parâmetros informados.

O *layout* da tela será modificado conforme proposto, mas manterá o mesmo *design* do portal. Dentro do projeto atual a *View Search* terá seu código fonte alterado para anteder a solução. A *DefaulController* terá funções modificadas e, se necessário, outras novas serão criadas.

4.1.3.1 Testes

O primeiro critério avaliativo corresponde ao funcionamento da parte lógica da tela, onde os campos preenchidos e selecionados devem ser transmitidos pelos métodos GET ou POST para o novo mecanismo de consulta.

Serão descritos testes nos quais serão simuladas pesquisas através do preenchimento dos campos, seleção de colunas e número de linhas. Cada simulação terá que ser documentada e tais informação terão que ser apresentadas ao

mecanismo de consulta sem perda ou distorção dos parâmetros informados pelo usuário.

Como segundo critério será avaliado a funcionalidade dos botões de adição e subtração de parâmetros. Um campo removido ou um campo adicionado em branco não deve ser enviado para o mecanismo de consulta.

O terceiro critério trada dos campos com auto complemento e dos *combo box* com textos pré-definidos. Realizar testes comparativos, onde as informações de tais campos terão de coincidir com consultas SQL criadas para validá-los.

4.1.4 Remodelagem da tela de resultados da pesquisa

4.1.4.1 Problema

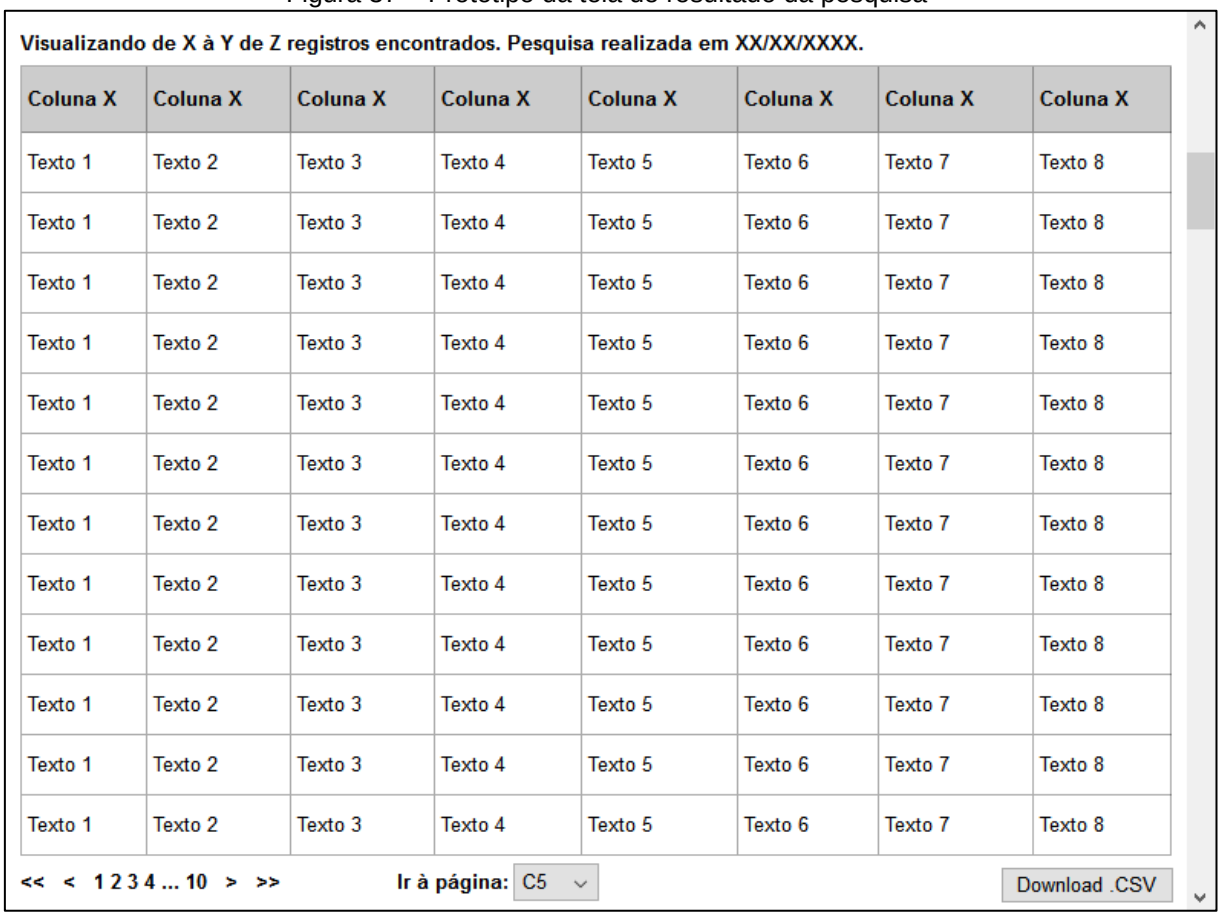
A tela atual reproduz os resultados exibindo um único um item por página. Isso torna trabalhosa e lenta a comparação entre os itens. A navegação entre os itens, como mostrado na Figura 27, permite alternar entre o primeiro, o último e os registros anteriores e seguintes ao que está sendo exibido atualmente. Não é possível ir diretamente para outro registro que não esteja visível.

Outro aspecto é o tamanho do *frame* da tela, que é estático, ou seja, ele não é redimensionável e nem adaptável a diferentes tamanhos de monitores.

4.1.4.2 Proposta

O resultado da pesquisa será exibido em uma nova janela ou aba, isso dependerá da configuração do *browser* do usuário, e não utilizará o *layout* da tela atual. Seu tamanho não será estático, utilizará todo o espaço disponível da janela e será redimensionável assim como a janela de *browser*. O protótipo da tela de resultado está ilustrado na Figura 37.

Figura 37 – Protótipo da tela de resultado da pesquisa



Fonte: Da Rosa (2015).

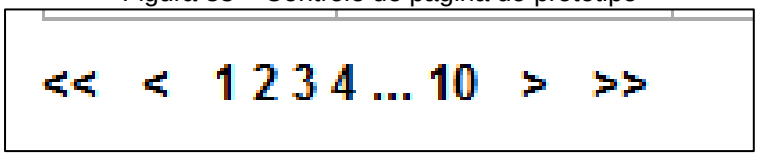
No topo da tela será exibido o número total de registros encontrados pelo mecanismo de busca, bem como a fatia atual que está sendo exibida (registro X à Y). Isso permitirá ao usuário localizar-se durante a leitura dos dados e ter uma referência do volume de dados apresentados à ele.

A data de realização da pesquisa também orientará o utilizador quanto à validade dos dados, pois trata-se de uma base dinâmica e com constantes atualizações e inserções. Os resultados podem variar de um dia a outro.

A tabela será paginada e cada paginação possuirá o número de linhas informadas pelo utilizador na tela de pesquisa. Cada linha corresponde a um item distinto dos outros. O *layout* da tabela será simples e limpo: fundo branco, linhas finas e pretas, e cabeçalhos destacados das demais linhas. Para tanto, será utilizado HTML, *JavaScript* e CSS para a formação.

O controle de paginação está localizado na parte inferior do protótipo e destacado na Figura 38. Ele permitirá avançar entre a primeira, a última e as páginas intermediárias à atual.

Figura 38 – Controle de página do protótipo



Fonte: Da Rosa (2015).

Esse controlador não é diferente do atual, mas é necessário pois é fácil utilização comparando-o com o campo C5, controlador de paginação, conforme Figura 37. Com um clique ele permite avançar e retornar para páginas vizinhas.

O campo C5 listará todas as páginas disponíveis e permitirá que seja selecionada a página de destino, sem precisar passar por páginas intermediárias.

O botão “Download .CSV” irá acionar uma função na *Controller* que irá criar o arquivo e iniciará o download do mesmo.

Para tanto, será criado uma *Controller* que receberá os parâmetros pelos métodos GET ou POST. Essa *Controller* possuirá métodos que chamarão as funções que criarão as *querys* de SQL, executarão a consultado no banco de dados e enviam o resultado para *View* montar a tela.

Uma *View* nova também será criada desvinculado do *layout* principal do projeto, permitindo assim que o design proposto seja feito.

4.1.4.3 Testes

Para verificar a eficácia da solução, deverão ser testados os seguintes tópicos:

- verificar se os parâmetros informados e envidados pela tela de pesquisa conferem com os recebidos na *Controller*;
- verificar se a função que gera a query está agregando todos os parâmetros à pesquisa;
- verificar se a *query* executa pelo programa retorna, em tela, os mesmos resultados que seriam obtidos caso a *query* fosse executada no SGBD.

4.1.5 Formato de arquivo para download

4.1.5.1 Problema

O portal disponibiliza o download das informações em três formatos de arquivos textos distintos: TXT, XML e CSV. Apesar da diversidade, os formatos atuais exigem que o conteúdo siga padrões distintos. Formatos como o TXT e XML estão perdendo popularidade. O TXT trata-se de um formato livre definição e padrões, ficando a cargo do criador escolher tais métricas, enquanto o XML possui padrões que tornam complexa na sua construção e leitura devido as *tags* (rótulos).

4.1.5.2 Proposta

O *download* do resultado da pesquisa será disponibilizado em um único formato de arquivo texto: o .CSV. O arquivo terá o mesmo conteúdo resultante da pesquisa incluindo os dados paginados, ou seja, os dados não visíveis. As colunas serão separadas por tabulação, conforme Figura 39.

Figura 39 – Protótipo do arquivo CSV

Coluna X	Coluna X	Coluna X	Coluna X	Coluna X	Coluna X	Coluna X	Coluna X
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8
Texto 1	Texto 2	Texto 3	Texto 4	Texto 5	Texto 6	Texto 7	Texto 8

Fonte: Da Rosa (2015).

A *Controller* que será criada para realizar o resultado em tela já possuía a consulta e a instância de conexão com o banco de dados. Uma função nova será

adicionado a essa *Controller* que irá montar o arquivo .CSV e será o gatilho que iniciará o *download*.

4.1.5.3 Testes

Para validar o arquivo, os seguintes testes terão de ser realizados:

a) o conteúdo deverá ter o número de linhas igual ao número de registros, exceto pela primeira (cabeçalho);
b) não deve haver linhas repetidas;
c) executar a consulta que gerou a consulta no SGBD e comparar o resultado com o arquivo gerado.

4.2 CONSIDERAÇÕES FINAIS

Com melhoria de software proposta nesse capítulo, o banco de dados PromotoresDB obterá maior desempenho para retorno de consultas. O novo modelo proposto é amplamente usado em solução de BI (*Business Intelligence*) utilizada pela TI a fim propor agilidade e resultado nas tomadas de decisões de grades a pequenas corporações e empresas.

O portal IntergenicDB ganhará uma nova ferramenta de busca que permitirá maior diversidade e velocidade nas pesquisas. Tanto a tela de pesquisa quando a de resultado estarão adequadas ao novo modelo proposto para o PromotoresDB, tendo os mesmos benefícios agregados.

5 DESENVOLVIMENTO

Neste capítulo serão discriminadas, sucintamente, as etapas realizadas para atingir as soluções propostas no capitulo 4. Inicia-se pela criação do DW, que consiste em converter as *views* Family e Kingdom, em tabelas, criar o *script* SQL que irá gerar a tabela fato, e criar as consultas para medir os tempos de execução e compará-los na fase de testes.

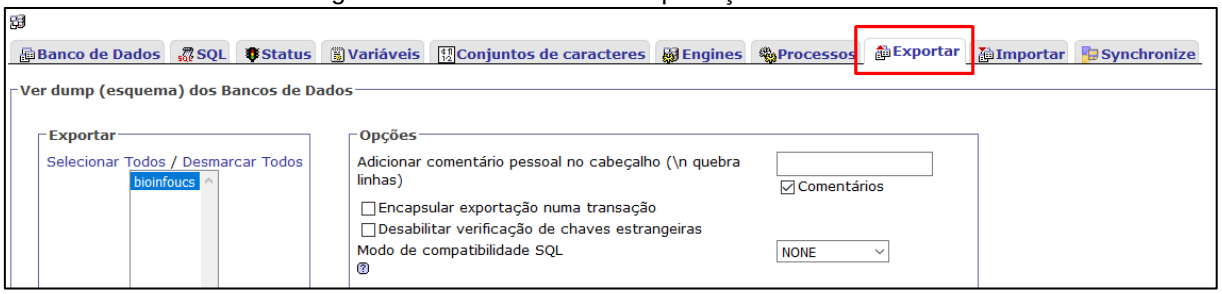
Após, será desenvolvido novo mecanismo de pesquisa e suas respectivas telas de consulta, resultado, acesso ao banco de dados e download do resultado, voltados a implementação do *Data Warehouse*.

5.1 AMBIENTE DE DESENVOLVIMENTO E TESTES

Visando não interferir em trabalhos paralelos, não danificar a base de dados do servidor e ter a disponibilidade de uma base de dados local, foi criada uma VM (*Virtual Machine* ou Máquina Virtual) com a seguinte configuração: Sistema Operacional Centos 7 (Linux); 8GB de memória RAM; 50GB de disco; 2 processadores de 2 *cores* com 2.30GHz; MySQL Server.

Para essa base de testes, foi exportada toda a estrutura (*schema* ou esquema*)* e os dados do banco atualmente disponíveis no servidor. A ferramenta utilizada está ilustrada na Figura 40, que exibe a guia onde encontra-se tal recurso.

Figura 40 – Ferramenta de exportação do servidor



Fonte: DA ROSA (2006).

5.1.1 Criação do *Data Warehouse*

Nesta sessão, serão apresentadas as etapas para a criação da tabela fato e demais modificações propostas na sessão 4.1.1, além de ajustes na base de dados para a consulta.

5.1.1.1 Converter as *views* em tabelas

O primeiro ato da conversão das *views* Kingdom e Family para tabelas é a criação de uma consulta que identifique os diferentes nomes de famílias e reinos. Tal informação está cadastrada na tabela Organism. Uma vez identificadas as diferentes famílias e reinos com uma consulta[43] (destacado na Figura 40), pode-se adaptar o *script* para criar as tabelas (Figura 41). Após, é ajustado o recurso de auto incremento dos campos FamilyID e KingdomID para seguirem a sequência correta.

Figura 41 – Script para criar as tabelas

```
CREATE TABLE `Family_tmp` (
    FamilyID INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY
) AS SELECT DISTINCT (
    Family
) AS Name
FROM `Organism`;

CREATE TABLE `Kingdom_tmp` (
    KingdomID INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY
) AS SELECT DISTINCT (
    Kingdom
) AS Name
FROM `Organism`;
```

Fonte: Da Rosa(2016).

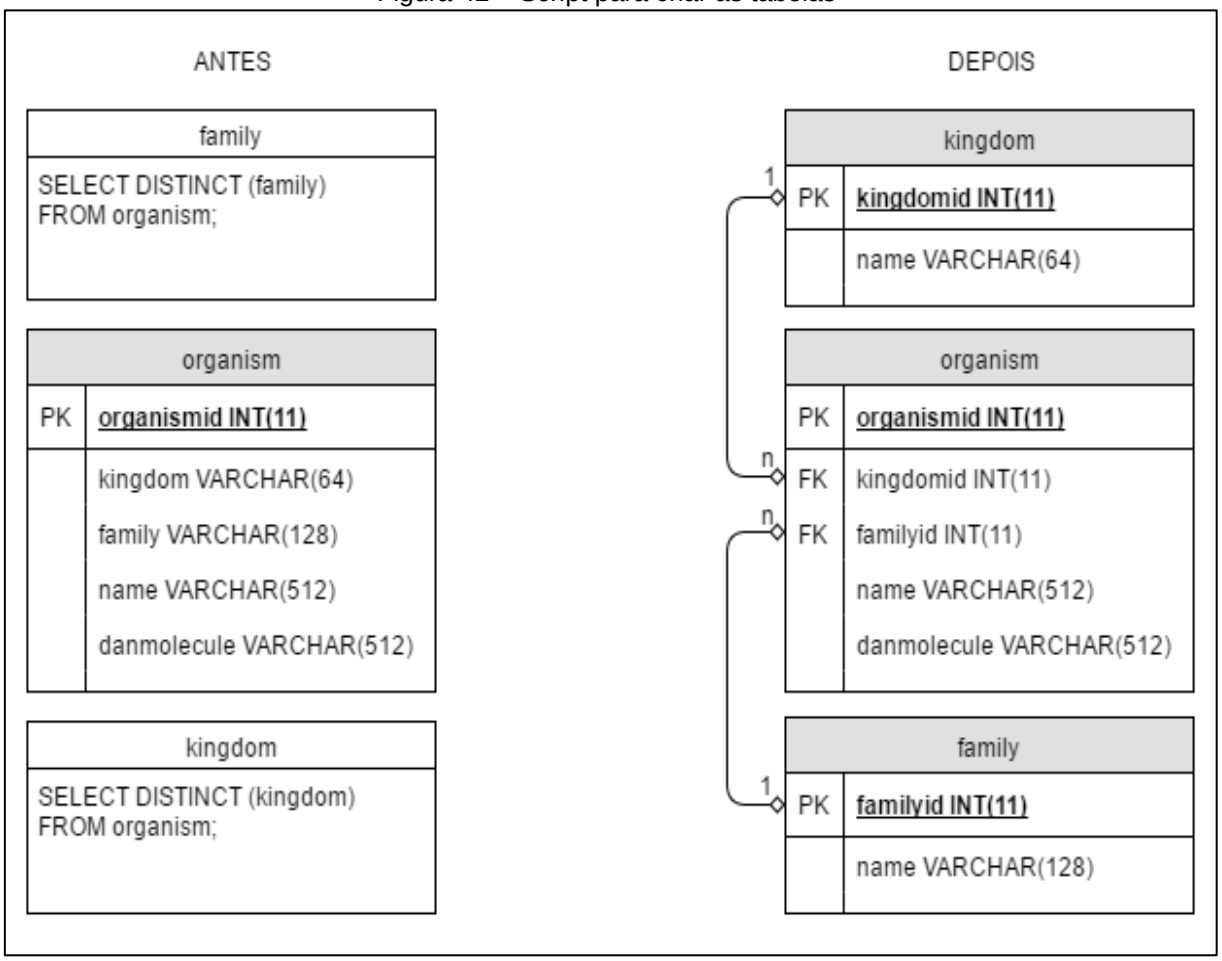
A última etapa dessa sessão, será a execução de ajustes na tabela Organism, a exclusão das *views* e renomear as tabelas criadas anteriormente. Até então, a tabela

[43] O comando *CREATE TABLE AS SELECT* permite criar tabelas e inserir dados simultaneamente a partir de uma consulta com o comando *SELECT*. Disponível em: <http://dev.mysql.com/doc/refman/5.7/en/create-table-select.html>. Acessado em: 25/10/2016

Organism (modelo ER na Figura 42) possui a coluna com os nomes que será substituída pelas colunas com identificação através de chave estrangeira. Para isso:

a) adiciona-se, as colunas KingdomID e FamilyID, na tabela Organism;
b) atualizar as linhas para que as novas colunas recebam os números de identificação existentes nas tabelas criadas, tendo como condição que *Organism.Family = Family_tmp.Name* e *Organism.Family = Kingdom _tmp.Name*;
c) eliminar as *views*;
d) renomear a tabela *Family_tmp* para *Family* e *Kingdom_tmp* para *Kingdom*;
e) alterar os novos campos da tabela *Organism*, tornando-os chaves estrangeiras das suas respectivas tabelas;
f) remover os campos *Family* e *Kingdom* da tabela Organism.

Figura 42 – Script para criar as tabelas

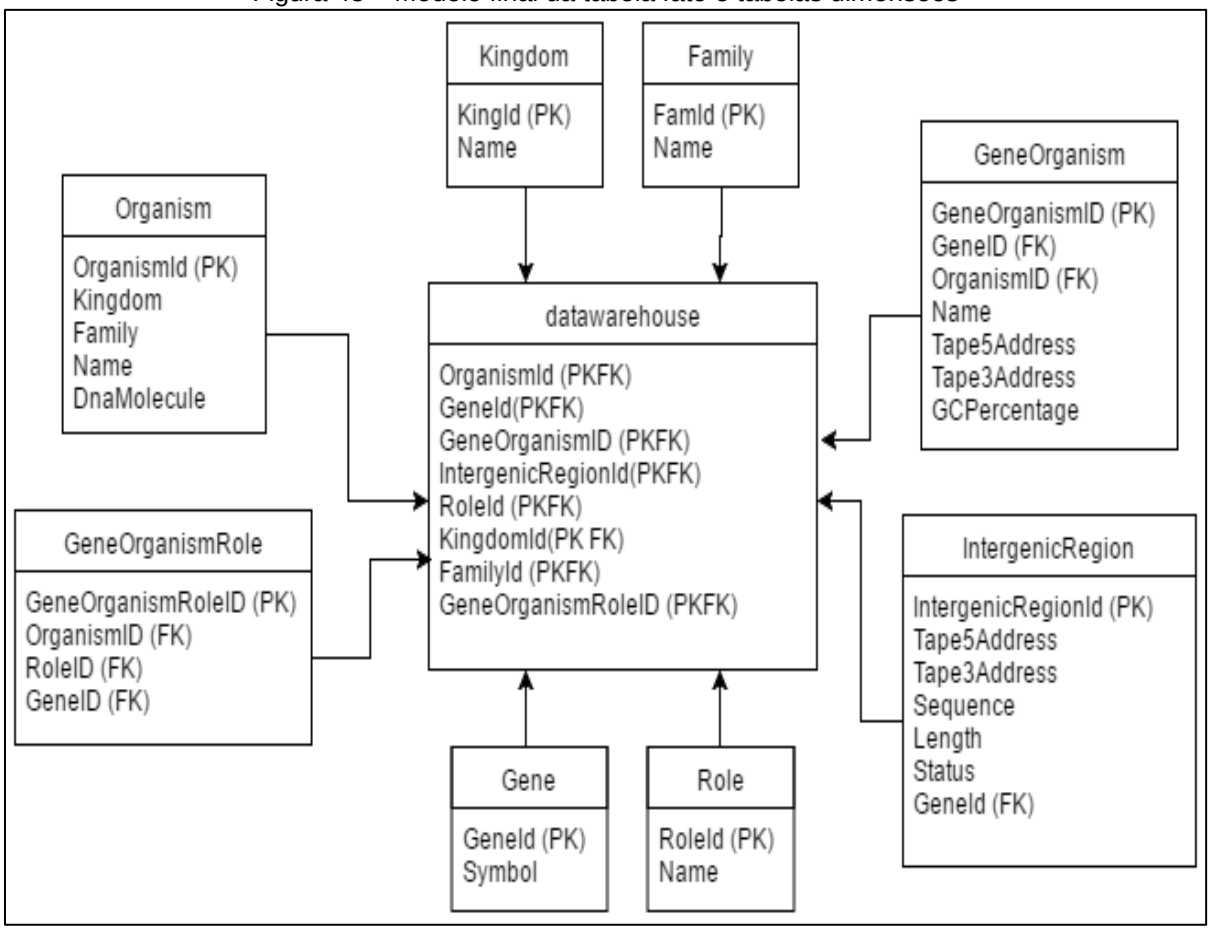


Fonte: DA ROSA (2016).

5.1.1.2 Criar a tabela fato

A tabela fato será chamada de *datawarehouse* e é formada, principalmente, pelos campos de *ids* das tabelas dimensões, mostrada na Figura 43:

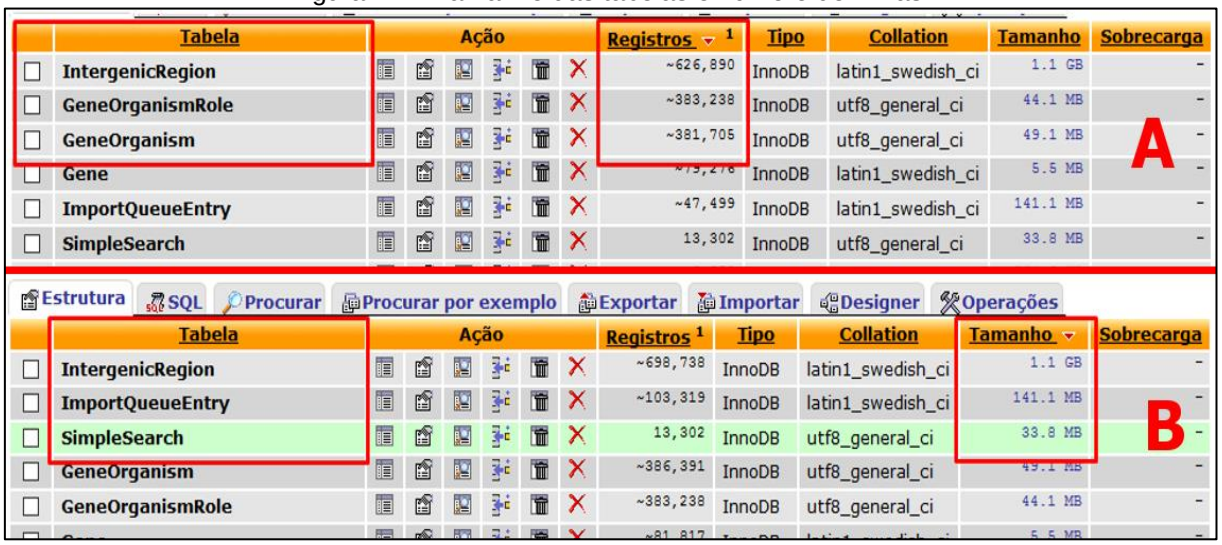
Figura 43 – Modelo final da tabela fato e tabelas dimensões



Fonte: DA ROSA (2016).

Para determinar a melhor estratégia de criação da tabela fato, inicia-se selecionando as maiores tabelas da base. As tabelas *IntergenicRegion*, *GeneOrganism* e a *GeneOrganismRole*, mostradas na Figura 44, que apresentam o maior número de linhas (Figura 44 A) e os maiores volumes (Figura 44 B). Com elas, os testes e simulações fornecerão valores de tempos mais próximos da realidade ao executar os testes e criar o *script* (sequência de comandos) que irá gerar a tabela fato.

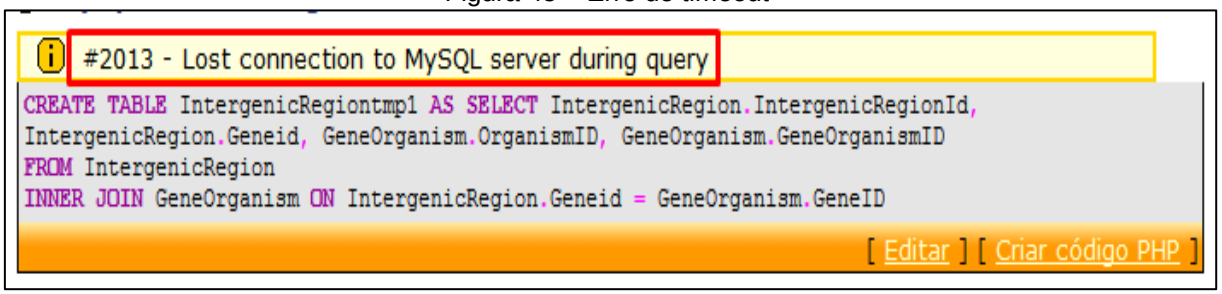
Figura 44 – Tamanho das tabelas e número de linhas



Fonte: Da Rosa (2016).

Ao executar as primeiras consultas na base de teste e no servidor, o programa informou erros de *timeout*. Mesmo utilizando outras alternativas para unir[44] as tabelas, buscando otimizar a consulta, o problema persistiu. Um exemplo dos erros está na Figura 45, onde verifica-se um exemplo das consultas utilizadas.

Figura 45 – Erro de timeout

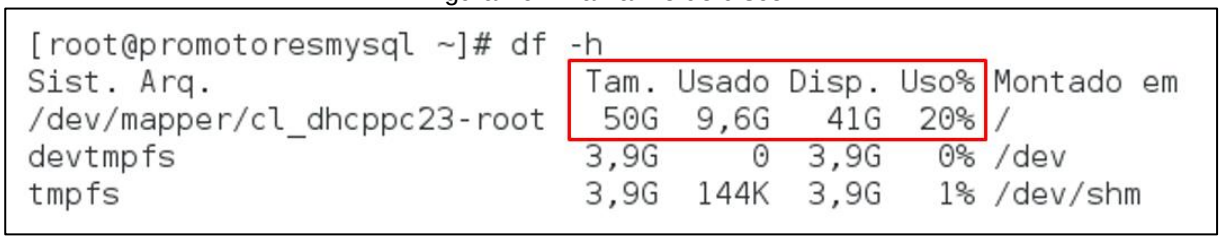


Fonte: Da Rosa (2016).

Mesmo realizando ajustes nas configurações do banco de dados, que permitam a execução de *consultas* por um maior intervalo de tempo, teve-se como consequência, problema de falta de espaço no disco. Conforme visto na Figura 46, a VM de teste possui 50GB de espaço, sendo que somente 20% está em uso antes da execução da query.

[44] A melhor abordagem para unir as tabelas é a utilização dos comandos *JOIN*. Esses comandos para unir tabelas são recomentados pela documentação e, nesse caso em especifico, utiliza-se o comando *INNER JOIN*. Disponível em: <http://dev.mysql.com/doc/refman/5.7/en/join.html>. Acessado em: 11/11/2016.

Figura 46 – Tamanho do disco.



Fonte: Da Rosa (2016);

Durante a execução da consulta, após ajustar os parâmetros de tempo, o espaço livre foi diminuindo, gradualmente, chegando a ocupar 99% do espaço total. A contramedida foi aumentar o tamanho do disco para 100GB, mas o problema reincidiu. No servidor não é possível ajustar os parâmetros de tempo desejáveis e o tamanho máximo disponível é de 4GB para MySQL.

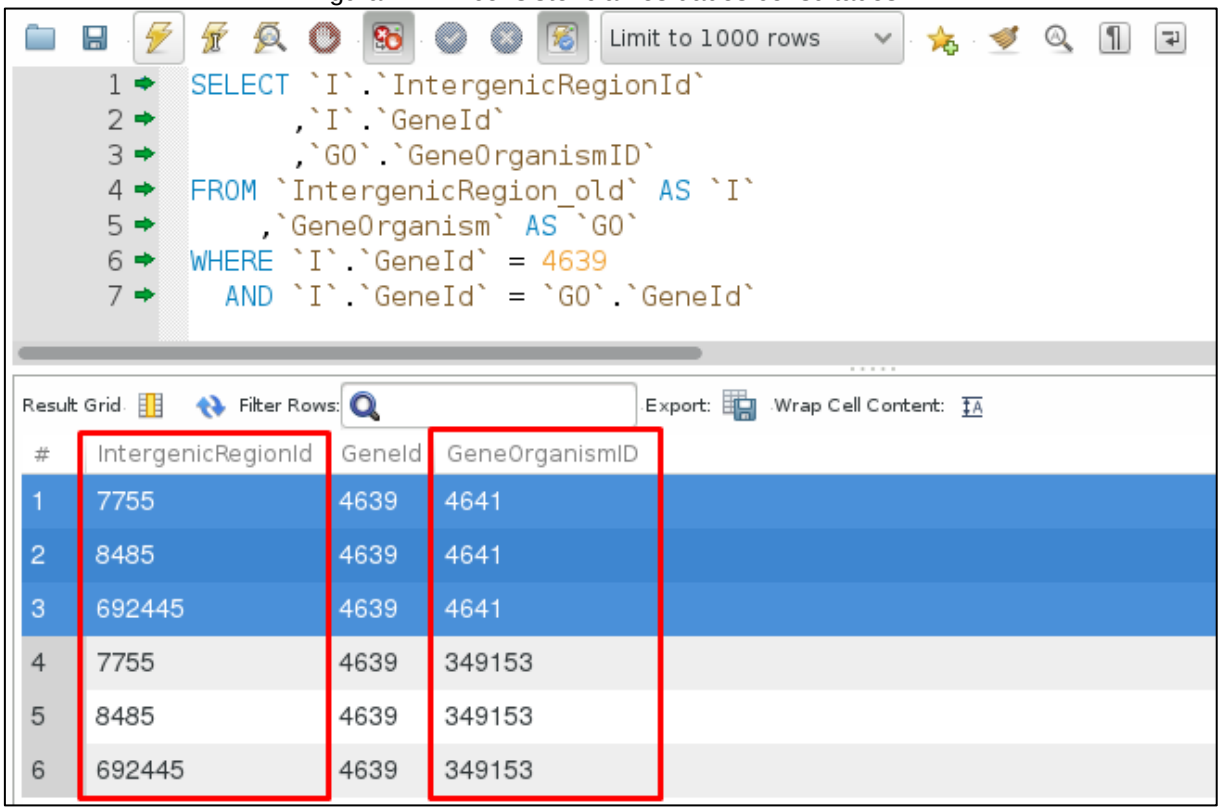
A base de dados atual, no servidor e local, ocupam 1.5GB aproximadamente. Isso significa que a consulta fez o banco de dados local crescer aproximadamente 66 vezes, crescimento incomum para uma simples combinação entre duas tabelas, onde a primeira possui 1.1GB e, a outra, menos de 50MB.

Ao executar mais testes e olhando mais atentamente a consulta, o resultado e os recursos do programa MySQL Workbench, na base local e na do servidor, foi encontrado um resultado inesperado, caracterizando uma inconsistência nos relacionamentos das tabelas.

5.1.1.2.1 Inconsistência nos relacionamentos entre tabelas

Tal inconsistência está em destaque na Figura 47, na qual vê-se também a consulta de teste e um limitador para exibir somente 1000 linhas. Verifica-se, no destaque, que *os GeneOrganism.GeneOrganismID* estão vinculados em mais de *um Intergenic.IntergenicId*, ou seja, a mesma região intergênica está sendo vinculada a diferentes organismos.

Figura 47 – Inconsistência nos dados consultados.



Fonte: Da Rosa (2016).

A *GeneOrganism* é uma tabela de relacionamento, vinculação entre um gene cadastrado na tabela *Gene* e seu organismo, cadastrado na tabela *Organism*. Segundo o usuário, uma região intergênica pertence unicamente a um organismo. Para atender a essa regra, os campos que interligam a *GeneOrganism* e *IntergenicRegion*, é o *GeneOrganismID* (Figura 47).

A base de dados, atualmente, está realizando esse relacionamento através do campo *GeneID*, causando a inconsistência. Isso faz com que uma região intergênica (*IntergenicRegionID)* tenha mais de um organismo (*GeneOrganismID)*

Problema similar também ocorre na tabela *GeneOrganismRole.* Sua função é similar à *GeneOrganism*, onde o objetivo é vincular um organismo a uma regra (*role*) do gene.

Com esses fatos, constata-se que a base de dados atual não está integra. Os relacionamentos criados entre as tabelas estão incoerentes, causando duplicidades, pois o campo *GeneID* não é opção para tal fim.

5.1.1.2.2 Alterações nas tabelas para corrigir as inconsistências

Mesmo com estes problemas, pode-se trabalhar para corrigi-los e continuar a desenvolver a tabela fato, apesar dos dados não estarem se relacionando corretamente. O que requer refazer a carga[45] de dados, serão utilizados mesmo assim, pois o volume de dados também é importante para os testes.

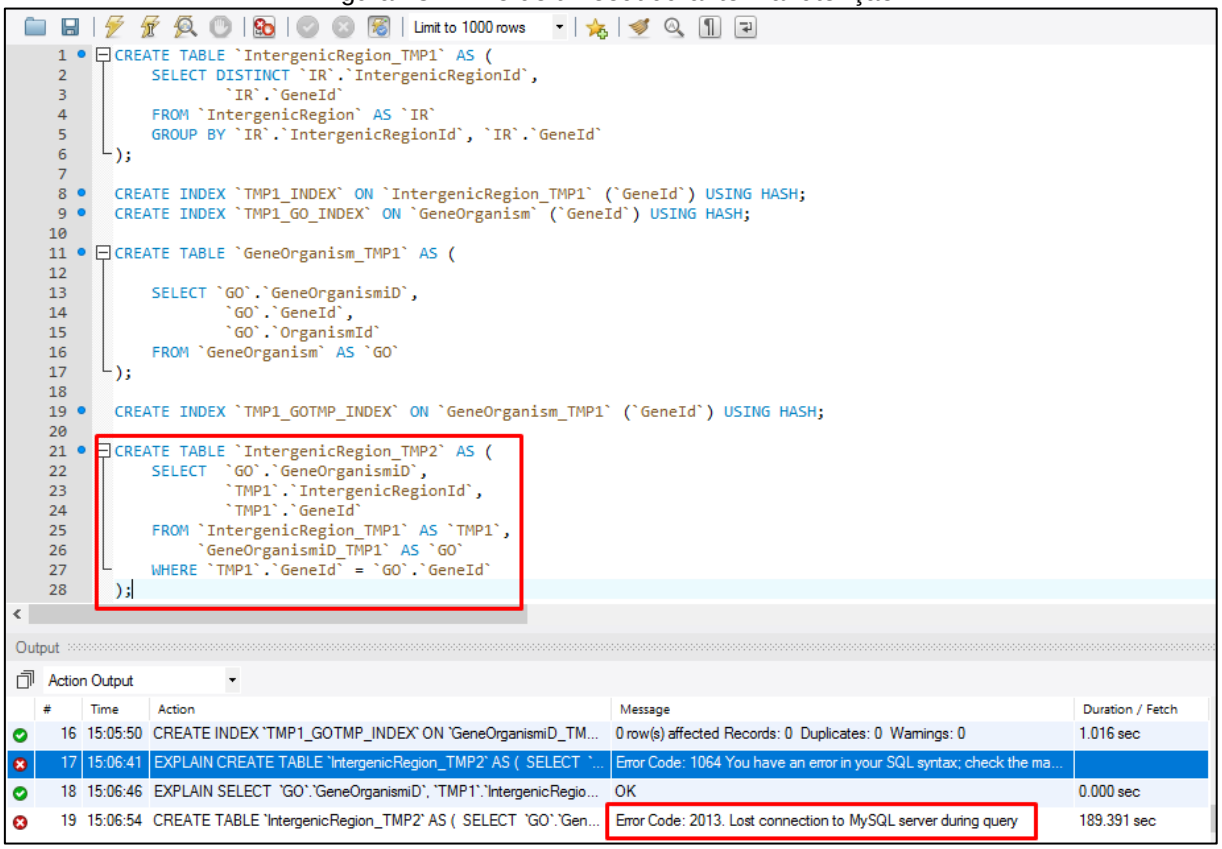
Para obter volume de dados e consistência entre eles, os passos a seguir serão efetuados nas *IntergenicRegion* e *GeneOrganismRole*:

a) eliminar as inconsistências, removendo os dados duplicados;
b) adicionar um novo campo, com o nome *GeneOrganismID*;
c) povoar o novo campo com valores da *GeneOrganism.GeneOrganismID*, através das atuais chaves estrangeiras existentes nas tabelas;
d) criar um relacionamento entre o novo campo *IntergenicRegion*.*GeneOrganismID* e o GeneOrganism.GeneOrganismID, através de chave estrangeira;
e) eliminar a coluna *IntergenicRegion*.*GeneID*.

Baseado em técnicas para *tuning* de um banco de dados, foram utilizados os comandos *JOIN*, *CREATE TABLE AS, SET SESSION* e outros, para sanar os problemas de *timeout* – tanto no servidor como na VM. Mesmo com essas contramedidas realizadas, os problemas prevaleceram. Como mostrado na Figura 48, os comandos que têm como objetivo criar a tabela fato *datawarehouse*. O trecho de código em destaque mostra a união entre duas tabelas através do gene e logo abaixo, também em destaque, o erro de *timeout* durante a execução.

[45] A carga de dados será realizada pela equipe do projeto IntergenicDB2.0.

Figura 48 – Erro de timeout durante manutenção



Fonte: Da Rosa (2016).

Numa reunião[46] com o usuário foi apresentado e discutido, entre outros assuntos, as limitações e problemas do bando de dados atual. Como alternativa o usuário optou por continuar com o mesmo provedor e trocar o SGBD MySQL para o PostgreSQL (PG). Assim necessitando realizar uma migração da estrutura e dos dados.

5.1.1.2.3 Migração do banco de dados

O processo de migração adotado[47] foi o ETL (*Extact, Transform e Load*). Os passos, basicamente, são extrair a estrutura e os dados da base atual, realizar as modificações (corrigir as inconsistências), operações necessárias para a conversão do SQL do MySQL em PostgreSQL e carregar para a nova base.

[46] Reunião realizada no dia 5 de agosto de 2016 com o usuário e a equipe do projeto Intergenic2.0.
[47] Definido pela equipe do projeto IntergenicDB2.0,

A extração foi executada utilizando as ferramentas do servidor, já mostrada na Figura 40. O arquivo resultante é um script em linguagem SQL compatível com o MySQL.

Existem diferenças na linguagem SQL entre distintos fabricantes. Para adaptarmos o *script*, foi utilizada a documentação oficial dos bancos de dados MySQL[48] e PostgreSQL[49]. Essa pesquisa resultou na Tabela 1, a qual é um resumo das principais modificações a serem feitas no *script*.

Tabela 1 – Converter Comandos do script de MySQL para PostgreSQL

De MySQL	Para PostgreSQL
unsigned	Remover em todo o script
int(x)	*int* ou *bigint* (de acordo com a tabela)
int(x) *AUTO_INCREMENT*	*bigserial*
KEY nome_index (campo)	/*colocar depois do *CREATE TABLE*, da tabela*/ *CREATE INDEX* nome_index *ON* nome_tabela *USING BTREE* (campo);
tinyint(x)	*smallint*
PRIMARY KEY (campo)	*CONSTRAINT* nome_constraint *PRIMARY KEY* (campo)
float	*double precision*
UNIQUE KEY nome_constraint (campo)	*CONSTRAINT* nome_constraint *UNIQUE* (campo)
mediumtext	*text*
char(x)	*character*(x)

Fonte: Da Rosa (2016).

Dentre as diferenças verificadas, o tipo *bigserial* do PostgreSQL chama a atenção pela simplicidade. Ao definir um campo com esse formato, automaticamente ele cria uma *SEQUENCE* (função de auto incremento do PostgreSQL) e define o campo como sendo do tipo *bigint*. De modo similar ao *AUTO_INCREMENTE* do MySQL, ele atribui um nome à sequência (geralmente criada a partir do nome do campo e da tabela) e o parametriza para incrementar uma unidade por vez (o padrão mais utilizado).

[48] Disponível em: <https://dev.mysql.com/doc/workbench/en/wb-migration-database-postgresql-typemapping.html>. Acessado em: 04/10/2016.

[49] Disponível em: <https://www.postgresql.org/docs/8.3/static/datatype.html>. Acessado em: 04/10/2016.

Além da Tabela 1, que trata dos especificadamente de comandos, foi criada uma tabela para palavras e delimitadores, as quais são reservadas para o sistema no PG e, é recomenda-se não as utilizar como nomes de campos, tabelas e outros. A Tabela 2, lista essas palavras e delimitadores onde a maioria delas serão colocadas entre aspas ou trocadas por sinônimos.

Tabela 2 – Palavras reservadas no PostgreSQL

De MySQL	Para PostgreSQL
admin	*“admin”*
name	*“name”*
user	*users*
password	*“password”*
connection	*“connection”*
sequence	*“sequence”*
` (crase)	*Remover todas as crases*

Fonte: Da Rosa (2016).

Além da conversão, ouras modificações serão necessárias como, por exemplo, a exclusão da tabela *SimpleSearch* que não será mais usada pelo mecanismo de pesquisa proposto nesse trabalho. Ela será substituída pela tabela fato, a *datawarehouse*.

Os ajustes para resolver o problema de inconsistência também serão inseridos no *script* final. A tabela *IntergenicRegion* receberá a coluna *GeneOrganismID* e será chave estrangeira de *GeneOrganism*.*GeneOrganismID*. O campo *GeneID* não será importado e, por essa razão, será necessária a exclusão e criação de índices, chaves estrangeiras, chaves primarias e outros.

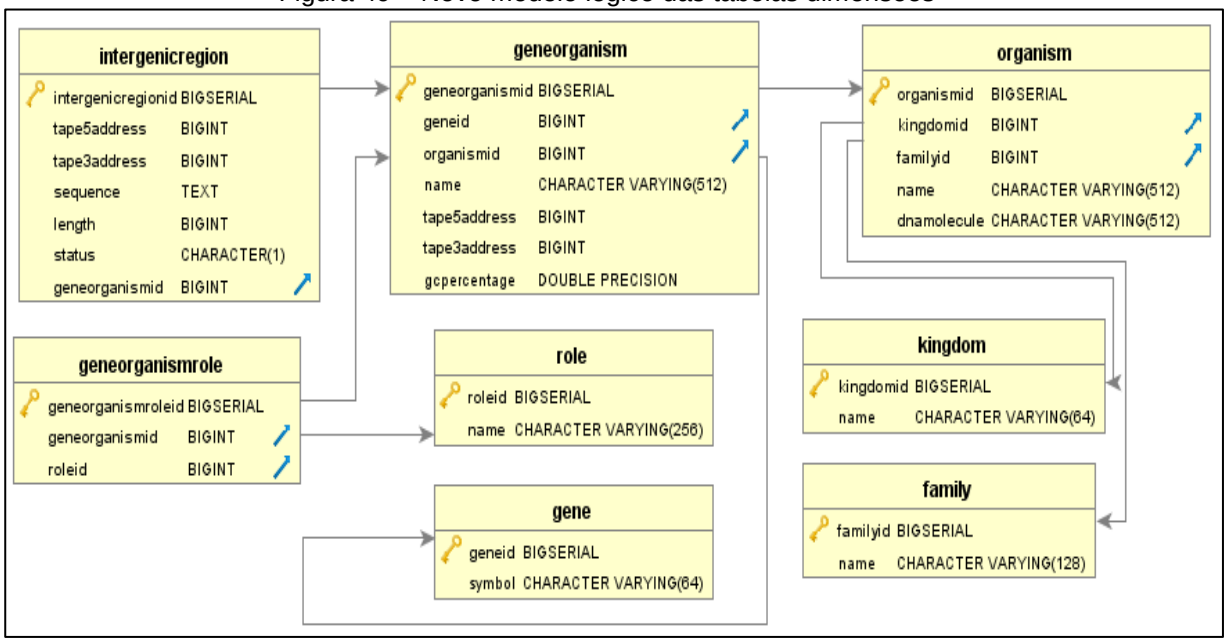
Da tabela *GeneOrganismRole*, não serão importados os campos *GeneID* e *OrganismID*. No lugar, será incluída a coluna *GeneOrganismID*, sendo chave estrangeira de *GeneOrganism*.*GeneOrganismID*. Também serão necessárias modificações na estrutura da tabela, como será feito para tabela *IntergenicRegion*.

O *script* final[50] resultante dessa atividade encontra-se no Anexo A deste trabalho. O *script* conterá essas e outras alterações que virão a ser mencionadas no decorrer do trabalho.

[50] Criado pela equipe do projeto IntergenicDB2.0.

Os dados da base atual, mesmo não tendo integridade, serão filtrados para remover os itens duplicados, exportados do MySQL para um arquivo texto, formatados manipulados para serem compatíveis com PostgreSQL e, por fim, importados para a base nova (PostgreSQL) para gerar um volume de dados e continuar com o desenvolvimento da proposta. Posteriormente, os dados, serão excluídos para receber a uma carga de dados integra. A Figura 49 exibe o novo modelo ER das tabelas dimensões após a migração e ajustes de inconsistências.

Figura 49 – Novo modelo lógico das tabelas dimensões



Fonte: Da Rosa (2016).

5.1.1.2.4 Criar a tabela fato no PostgreSQL

Para evitar problemas na nova base, uma cópia do esquema atual e das tabelas envolvidas será criada a partir de um *script* gerado por uma aplicação Java, chamada de PostgreInserts. Além de criar um ambiente seguro que proporcione a realização de testes sem causar conflitos com as demais atividades do projeto, a aplicação também gera dados aleatórios para povoar as tabelas.

A aplicação PostgreInserts gera *scripts* para criar um esquema temporário com as tabelas dimensões, seus índices, sequências e gera dados fictícios, porem íntegros em seu relacionamento, para povoar as tabelas. Não possui interface gráfica e para

volumes de dados diferentes deve-se alterar o código fonte e compilar novamente. As linhas do código fonte (Figura 50), são os parâmetros para gerar um script com maior ou menor volume de dados e linhas, mantendo a integridade.

Figura 50 – Parâmetros do programa PostgreInserts

```
98          //Parametrizaçao do script
99          //Local para salvar os arquivos
100         String diretorio = "D:\\";
101         //Numero de linhas
102         int geneRows = 625;
103         int familyRows = 100;
104         int kingdomRows = 25;
105         int roleRows = 1100;
106         int OrganismRows = 1600;
107         int GeneOrganismRoleRows = 1500000;
108         int IntergenicretionRows = 1500000;
109         //Tabela GeneOrganism, tera um numero de
110         //linhas <= geneRows * OrganismRows
111         Tables tb = new Tables("teste_med");
112         //Ocupar todos o espaço do campo.
113         //Se true e for um VARCHAR(512) por exemplo
114         //  , ira gerar um texto de 512 caracteres
115         //Se false o valor mínimo será de 5 caracteres e no máximo 512.
116         boolean espaco = true;
117         //Fim - Parametrizaçao do script
```

Fonte: Da Rosa (2016).

Com essa base de testes, pode-se definir quais dos tipos de índices terão melhor desempenho para otimizar as uniões (*JOIN*) entre as tabelas. Serão testados o tipo *hash* e *btree*. O tipo *hash*[51] somente trata comparações – se um elemento é exatamente igual ao outro - podendo apenas utilizar o operador de igualdade[52] (=) e não possui recursos de ordenação. A partir de uma operação, ele gera um identificador exclusivo, o que exige espaço em disco. O PostgreSQL considera o método depreciado[53], mas muito útil em colunas com números. Não é recomendado para campos de texto.

Para todos os tipos de dados compatíveis com PG, pode-se utilizar *btree*[54]. Trata-se de uma árvore binária que permite utilizar diferentes estratégias com

[51] Disponível em: <https://www.postgresql.org/docs/8.3/static/xoper-optimization.html>. Acessado em 24/09/2016.
[52] Disponível em: <https://www.postgresql.org/docs/8.3/static/xindex.html>. Acessado em 24/09/2016.
[53] Não recomenda a utilização segundo a documentação. Disponível em: <https://www.postgresql.org/docs/8.3/static/indexes-types.html>. Acessado em 15/09/2016.
[54] Disponível em: <https://www.postgresql.org/docs/8.3/static/xindex.html>. Acessado em 24/09/2016.

operadores – cinco operadores (>, >=, =, =<e <) – e reordenar os dados em tempo de execução sem necessitar criar índices e tabelas temporárias em *cache*.

Outro recurso para testar é o *Planner Method* (PM)[55]. Esse recurso permite ao PostgreSQL, se habilitado, optar pelo melhor índice, considerando o que possuir menor custo, assim otimizando as operações. Por padrão, esse parâmetro está habilitado. Para desligar, utiliza-se o comando SQL "*SET enable_seqscan = off*", antes de fazer a consulta. Para ligar, basta substituir o *off* pelo *on*

Utilizando as duas maiores tabelas, IntergenicRegion (700 mil linhas, aproximadamente) e *GeneOrganismRole* (300 mil linhas, aproximadamente), foram executados testes. O problema desses testes é obter um tempo justo (em milissegundos), pois tal necessidade é atrapalhada pelo *cache* do banco.

O *cache* tem a função de armazenar, de maneira instantânea, os últimos dados utilizados pelo banco e conceder um rápido acesso a essas informações. Hipoteticamente falando, ao executar uma consulta A, seu resultado será armazenado no *cache* do SGBD. Ao executar novamente a consulta A, ou outra consulta similar, o resultado será mais rápido devido a esses dados estarem no *cache*. Para tanto, é necessária uma sequência ordenada de comandos (Figura 51, Quadro 1) e a reinicialização do serviço do banco (por garantia).

Figura 51 – Comandos para limpeza do cache e exemplo de teste

Quadro 1

```
export PGPASSWORD=PromotoresPG
export PGPASSFILE=/.pgpass
su -c "psql -q -d postgres -c 'DROP INDEX IF EXISTS testeindice1'" postgres
su -c "psql -q -d postgres -c 'DROP INDEX IF EXISTS testeindice2'" postgres
su -c "psql -q -d postgres -c 'CREATE INDEX testeindice1 ON intergenicregion USING hash (geneorganismid)'" postgres
su -c "psql -q -d postgres -c 'CREATE INDEX testeindice2 ON geneorganismrole USING hash (geneorganismid)'" postgres
su -c "psql -q -d postgres -f cleanCache.sql" postgres
sync; systemctl stop postgresql-9.5; echo 3 > /proc/sys/vm/drop_caches; systemctl start postgresql-9.5;
su -c "psql -d postgres -f query.sql" postgres
```

Quadro 2

```
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
DEALLOCATE ALL;
CLOSE ALL;
UNLISTEN *;
SELECT pg_advisory_unlock_all();
DISCARD ALL;
DROP INDEX  IF EXISTS "IndexTeste1";
DROP INDEX  IF EXISTS "IndexTeste2" ;
DROP TABLE IF EXISTS testeindex;
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
DEALLOCATE ALL;
CLOSE ALL;
UNLISTEN *;
SELECT pg_advisory_unlock_all();
DISCARD ALL;
```

Quadro 3

```
\timing on
SET enable_seqscan = off;
CREATE TABLE testeindex AS
SELECT  ir.intergenicregionid,
  ir.tape5address,
  ir.tape3address,
  ir.length,
  ir.status,
  ir.geneorganismid,
  gor.geneorganismroleid,
  gor.roleid
FROM    intergenicregion as ir
INNER JOIN geneorganismrole as gor
USING (geneorganismid);
```

Fonte: Da Rosa (2016).

[55] Disponível em: <https://www.postgresql.org/docs/8.3/static/runtime-config-query.html>. Acessado em 24/09/2016.

No quadro 2 da Figura 51, verifica-se a sequência de comandos SQL realizadas para limpar o *cache* atual, executados na linha 5 da Figura 51. Já na linha 6 da Figura 51, demonstra-se a sequência de quatro comandos do sistema operacional Linux CentOS, que irá parar o serviço do banco, limpar o *cache* do sistema operacional e iniciar o serviço do banco. Por sua vez, Figura 51 quadro 3, trata-se de um exemplo das consultas SQL usadas para medir os tempos. Os índices originais das tabelas *intergenicregion* e *geneorganismrole* foram excluídos, mas somente para a execução dos testes. No lugar, foram criados índices temporários conforme a necessidade dos testes, os quais são executados nas linhas de 1 a 4 da Figura 51.

Cada teste (Tabela 3) consiste nas combinações de tipos de índices (*hash,* ou *btree,* ou sem índice) e o *Planner Method* (habilitado ou desabilitado). Quando executado, o índice anterior será excluído, sendo criado o novo (Quadro 1, linhas de 1 a 5), antes da limpeza do cache do banco. Para cada situação da Tabela 3, o *script* do Quadro 3 da Figura 51, tende a ser adaptado. Foram executados em ambiente de teste, a partir da *console* e usando *shell script* (linguagem de programação para ambientes Linux), representado no Quadro 1 da Figura 51.

Tabela 3 – Resultado dos testes de desempenho dos índices

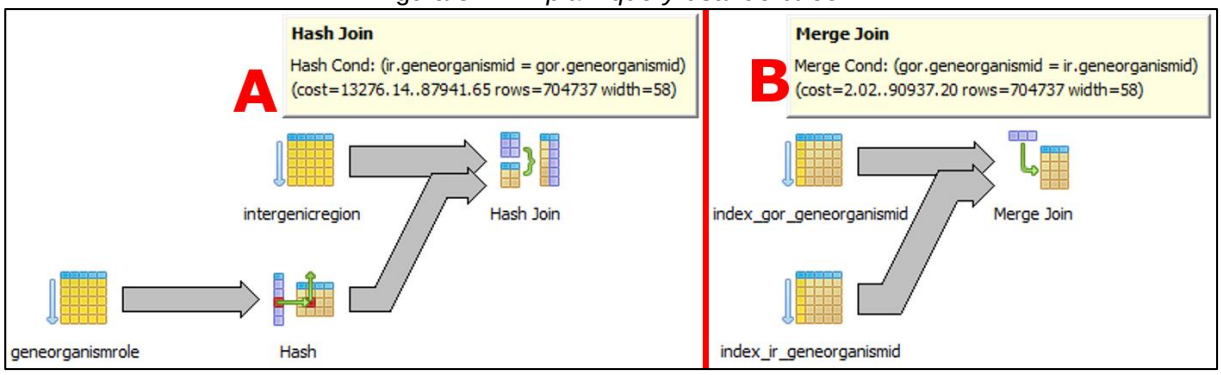
Teste	Índice	PM	Amostras de tempo em milissegundos				
			Tempo 1	Tempo 2	Tempo 3	Tempo 4	**Média**
3	btree	off	6908,815	6852,649	6853,058	6775,648	**6847,543**
4		on	7096,433	7007,074	6996,705	7052,148	**7038,090**
6	btree	on	7351,981	7390,321	7240,509	6937,701	**7230,128**
5	hash	on	9729,457	9805,725	10279,353	9757,585	**9893,030**
2	hash	off	13718,425	12696,612	12410,081	13110,388	**12983,877**
1		off	13068,319	13036,678	13291,680	12914,709	**13077,847**

Fonte: Da Rosa (2016).

Considerando que a linguagem SQL apresenta semelhanças entre MySQL e PG, foram adaptados os comandos desenvolvidos na sessão 5.1.2 para executá-los no PG. Durante a adaptação e o desenvolvimento do *script* que gera a tabela fato, foi possível utilizar o recurso *Explain Query*. Tal recurso, por meio de gráfico ou de texto, estima um custo, o número de linhas e o tamanho do custo que será exigido do SGBD para executar a consulta.

A Figura 52 mostra dois resultados distintos do *Explain Query* sob uma consulta SQL, ambos usando índice *btree.* A Figura 52 A está com o *Planner Method* ligado, enquanto a B está desligado. Verifica-se uma diferença no resultado do *Explain Query* dos testes de número 3 e 6 (Tabela 3). Essa diferença no custo inicial[56] que chega à ser 99.99% superior do item A (*cost*=13276.14) para o item B (*cost*=2.02). O custo final[57] de A (*cost*=879491.65) e B (*cost*=90937.20) possuem uma diferença pequena.

Figura 52 – *Explain query* usando *btree*



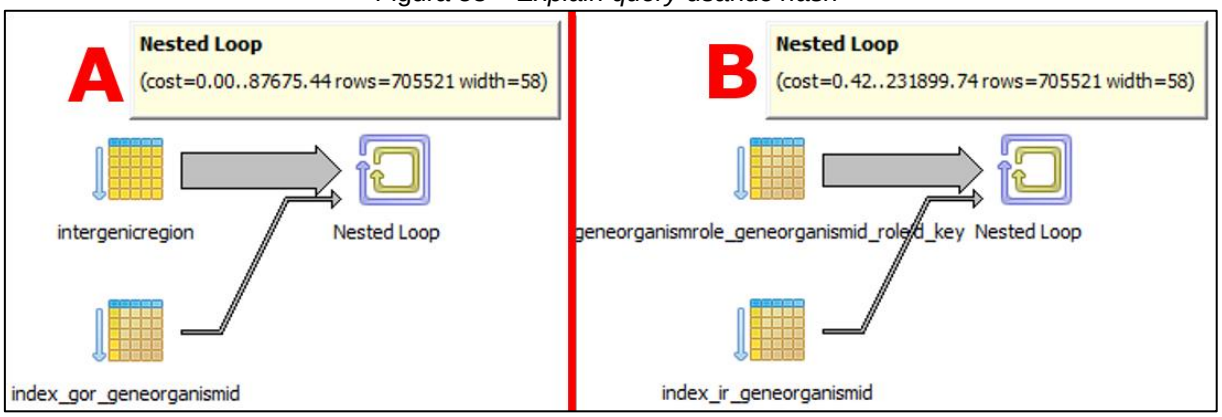
Fonte: Da Rosa (2016).

Pode-se observar na Figura 52 A (PM ligado) que o resultado foi unido através de *Hash join*[58], ao contrário da Figura 52 B, que utilizou os índices já criados. Isso significa que foi gerado um índice *hash* – cujo uso é desencorajado[59] pela própria documentação – para realizar essa união. O *Planner Method* busca encontrar a melhor estratégia, dentro do menor custo, mas não serão testadas todas as possiblidades necessariamente.

A Figura 53 mostra o resultado de dois testes com *Explain Query* (Tabela 3Tabela *3*, número 2 e 5), ambos usando *hash.* O A está com o PM ligado e o B, desligado. Ao ser comparado com a Figura 52, mostra um custo inferior, mas nos testes de tempos estão nas últimas posições na tabela. Quanto menor for o custo e o tempo, melhor desempenho a consulta terá.

[56] Custo inicial é o custo que a execução da consulta levará para exibir/retornar a primeira linha. Em casos de união de tabelas, deve-se priorizar o curso inicial ao analisar o *Explain*.
[57] Custo final é o custo que a execução da consulta levará para exibir/retornar todas as linhas. Quando deseja-se retornar todos os dados de uma consulta, sim o uso de limitadores, deve-se priorizar o custo final durante a análise do *Explain*
[58] Disponível em: <https://www.postgresql.org/docs/8.3/static/using-explain.html>. Acessado em 25/10/2016.
[59] Disponível em: < https://www.postgresql.org/docs/8.3/static/indexes-types.html>. Acessado em: 20/11/2016.

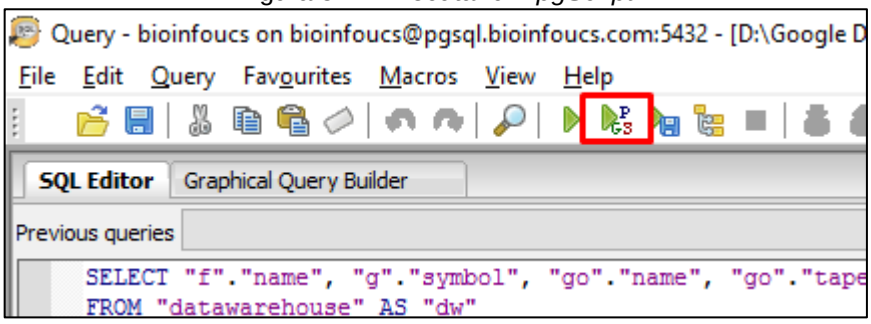
Figura 53 – *Explain query* usando *hash*



Fonte: Da Rosa (2016)

Com esses testes, podemos concluir que há uma demanda distinta para cada tipo de união. Será necessário atender caso a caso para criar um *script* com melhor tempo/beneficio, sendo que o aumento da base de dados e demais alterações poderão afetar o *script*, necessitando de ajustes. O *script* resultante dessa sessão, pode ser visto no Anexo II. Para executá-lo, utiliza-se o programa pgAdmin3[60], clicando no botão "*Execute pgScript*" mostrado na Figura 54.

Figura 54 – Executar um *pgScript*



Fonte: Da Rosa (2016).

5.1.1.3 Testes

O *datawarehouse* é uma tabela física onde os dados estão devidamente organizados, com seus índices prontos para acesso, enquanto a *view* necessita ser

[60] Disponível em: <https://www.pgadmin.org/>. Acessado em: 23/11/2016

montada em *cache* para depois estar disponível. Os comandos e as consultas realizadas nessa sessão de teste encontram-se no Anexo D.

Dentro da base de teste, existem três esquemas: *public* – a cópia da base atual, a *teste_med* e a *teste_grand* – gerado pelo programa *PostgreInserts*, contendo somente a estrutura, das tabelas dimensão e os dados gerados aleatoriamente. A Tabela 4 mostra os tamanhos dos esquemas e número de linhas envolvidos no teste.

Tabela 4 – Tamanhos e número de linhas das tabelas

	Esquemas					
	public		*teste_med*		*teste_grand*	
Tabelas	Nº de linhas	Tamanho (bytes)	Nº de linhas	Tamanho (bytes)	Nº de linhas	Tamanho (bytes)
Total	**225205**	**97075200**	**3093701**	**600186880**	**3094569**	**2353766400**
family	17	40960	100	73728	101	114688
gene	15095	1032192	625	147456	938	303104
geneorganism	61094	19218432	997776	194215936	997776	1407918080
geneorganismrole	75252	15671296	1094706	200220672	1094706	200220672
intergenicregion	61091	60096512	997769	204914688	997774	740777984
kingdom	1	40960	25	73728	25	73728
organism	88	90112	1600	344064	1599	3145728
role	12567	884736	1100	196608	1650	1212416

Fonte: Da Rosa (2016).

O programa *PostgreInserts* gerará vários arquivos *".sql"* que devem ser executados para criar um novo esquema com tabelas, índices, sequências e dados. Com isso, pode-se medir o tempo e ter uma ideia do desempenho do *script* que gera *datawarehouse* em bases de dados com maiores volumes de dados.

O número de linhas dos esquemas *teste_grand* e *teste_med*, é aproximadamente o mesmo. A diferença está na geração do esquema *teste_grand*, pois nele é utilizado o tamanho total dos campos de texto. Ou seja, se o campo aceita um texto de 512 caracteres, por exemplo, o valor inserido é de 512 caracteres.

O teste consiste em três execuções, em momentos distintos, do *script* do Anexo II nos três esquemas. Antes da criação de cada DW, serão executados os comandos e a reinicialização do banco de dados para limpar o *cache* como feito para obter os resultados da Tabela 3. Abaixo, na Tabela 5, seque o resultado obtido com os testes.

Tabela 5 – Tempos para a criação da DW

Esquemas	Tempo em minutos:segundos			
	Tempo 1	Tempo 2	Tempo 3	**Média**
public	00:36	00:21	00:22	**00:20**
teste_med	16:15	16:50	18:13	**12:49**
teste_grand	17:39	17:15	19:33	**13:37**

Fonte: Da Rosa (2016).

Analisando a média de tempo entre os esquemas, vê-se uma diferença de aproximadamente 14 minutos. A diferença é esperada, pois o número de linhas dos esquemas *teste_grand* e *teste_med* é 1374% maior. Outra observação é sobre a *teste_grand,* que possui um volume de dados 392% maior que a *teste_med*. Entretanto, isso se mostrou insignificante para o tempo de execução do teste de criação da DW.

O *script* possui um bom desempenho pois processou, comparou e uniu aproximadamente 3094569 linhas existente no esquema *teste_grand*. Não ocorreram problemas de *timeout* ou falta de espaço em disco durante a execução.

Para simular a antiga *SimpleSearch*, iremos criar uma *view* com lógica similar à da tabela *datawarehouse*. Isso se deve ao fato do atual banco em MySQL não estar funcionando adequadamente. A *view* irá utilizar os mesmos índices (das tabelas dimensões) criados para auxiliar a tabela fato, permitindo que as medidas de tempo sejam mais justas, já que ambas estarão sendo testadas nos mesmos esquemas. O código que cria a nova *SimpleSearch view* de teste, está contemplado no Anexo III.

Como primeiro teste, foi realizada uma consulta na *view* e na *datawarehouse*, requisitando todos os dados, ou seja, ver todas as colunas sem nenhum filtro. Os tempos estão na Tabela 6.

(continua)

Tabela 6 – Tempos do 1º teste

Esquemas	Origem	Tempo em milissegundos			
		Tempo 1	Tempo 2	Tempo 3	**Média individual**
public	*view*	119000	69000	97000	**95000**
	datawarehouse	375	219	250	**281,33**

(conclusão)

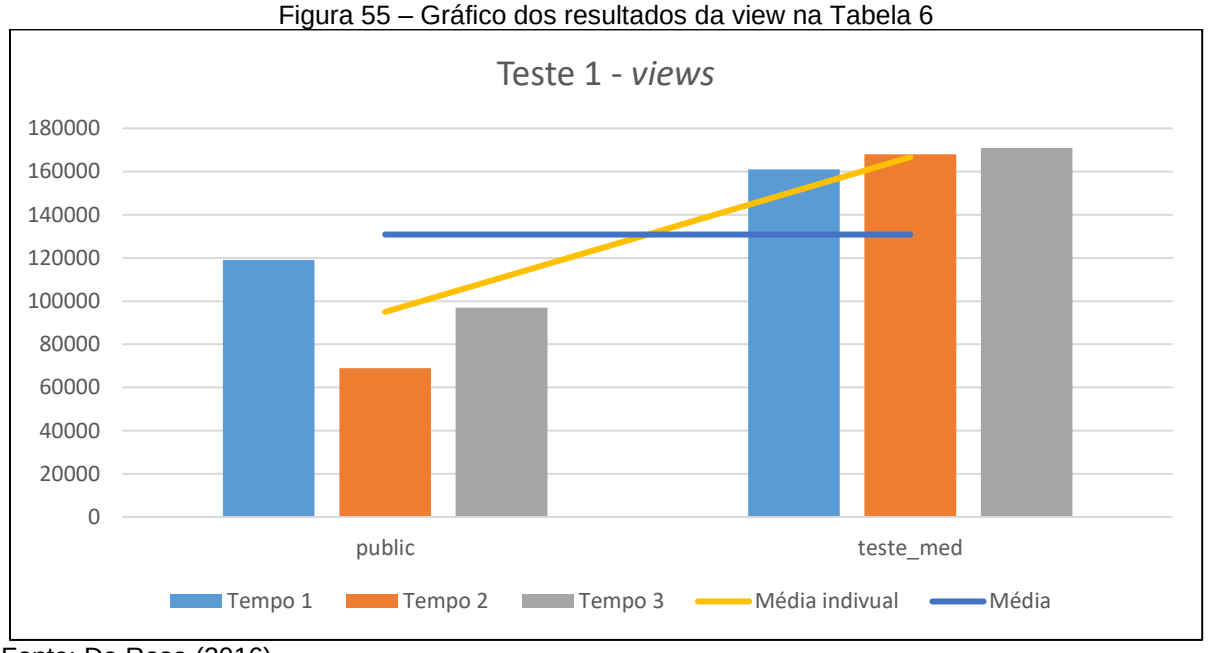
teste_med	*view*	161000	168000	171000	**166666,7**
	datawarehouse	94	78	98	**90**
teste_grand	*view*	*timeout*[61]	*timeout*	*timeout*	
	datawarehouse	*329*	*359*	*343*	**343,6667**

Fonte: Da Rosa (2016).

Devido às parametrizações de segurança do banco de dados existente nos três esquemas, não foi possível obter um tempo da consulta realizada na *view* do esquema *teste_grand*. Esse problema é resolvido utilizando um limitador para o número de linhas no resultado. Isso é viável utilizando os comandos SQL *LIMIT*[62] e *OFFSET* que serão implantados nas consultas do *datawarehouse*.

A Figura 55 mostra o crescimento do tempo da pesquisa de um esquema para o outro, sendo que o esquema *teste_grand* não entrou na análise pelo fato de ter ultrapassados os limites impostos pelo servidor.

Figura 55 – Gráfico dos resultados da view na Tabela 6



Fonte: Da Rosa (2016).

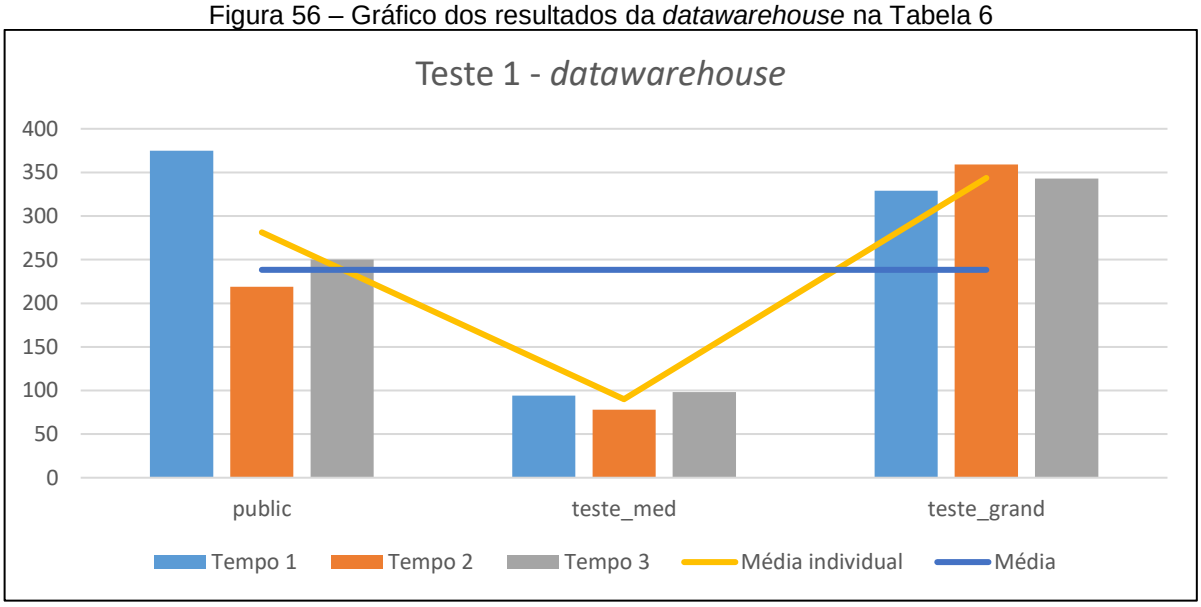
Enquanto na Figura 55 o maior tempo é de 3 minutos aproximadamente, na Figura 56 tem-se uma queda para 350 milissegundos. A vantagem da *datawarehouse* é a utilização dos parâmetros *LIMIT* e *OFFSET* que serão utilizados para a paginação

[61] O tempo máximo para a execução da consulta foi atingido, fazendo com que a consulta fosse cancelada. O tempo máximo é um parâmetro do banco de dados.

[62] No comando *LIMIT*, informa a quantidade de linhas (*n*) que se deseja visualizar. No *OFFSET*, informamos o número da linha a qual será a primeira a ser exibida das *n* linhas determinadas no comando *LIMIT*.

do resultado em tela para o usuário. Apesar do tamanho do esquema *public* ser menor que o esquema *teste_med*, aquele que apresentou os maiores tempos. O custo para processar a *teste_med*, ao executar o *Explain Query* é menor que a *public,* devido à quantidade de linhas que se relacionam. Mesmo assim, a diferença de tempo é mínima, em milissegundos.

Figura 56 – Gráfico dos resultados da *datawarehouse* na Tabela 6



Fonte: Da Rosa (2016).

Já no segundo teste, a consulta solicita todas as colunas, procurando por genes que contenham a letra "b"[63] e ordenando pelo nome dos genes resultantes. O resultado está na Tabela 7.

Tabela 7 – Tempos do 2º teste

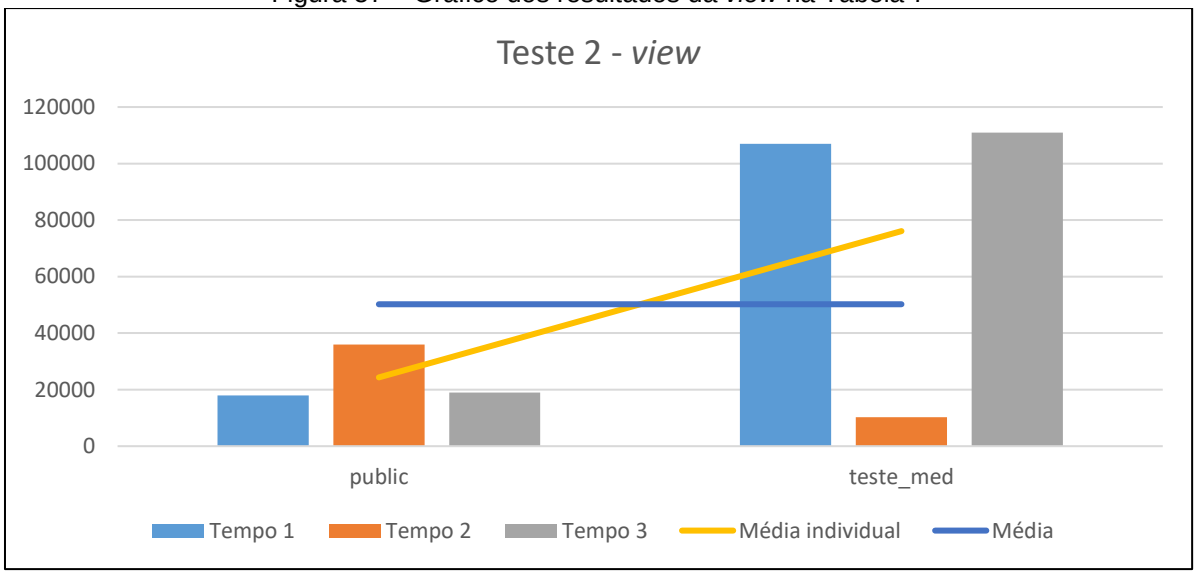
Esquemas	Origem	Tempo em milissegundos			
		Tempo 1	Tempo 2	Tempo 3	**Média individual**
public	*view*	18000	36000	19000	**24333,33**
	datawarehouse	1500	1500	1700	**1566,667**
teste_med	*view*	107000	10300	111000	**107000**
	datawarehouse	18400	19300	17100	**18266,67**
teste_grand	*view*	*out of memory*[64]	*out of memory*	*out of memory*	**nulo**
	datawarehouse	*75000*	*98800*	*96000*	**89933,33**

Fonte: Da Rosa (2016)

[63] O comando *ILIKE* é *case insensitive*, ou seja, não faz diferenciação entre maiúsculas e minúsculas.
[64] O volume de dados era muito superior aos recursos (hardware) disponíveis para exibi-los em tela.

Pode-se ver que o problema do primeiro teste na *view* se repete, mas desta vez por falta de memória para gerenciar os dados a serem exibidos em tela. Quando o volume de dados for significante, a *view* se mostra instável. Na Figura 57 no esquema *teste_med*, a segunda amostra de tempo é quase zero perto das outras. Esse comportamento aparenta ser resultado de uma pesquisa que obteve os dados da *cache*, mesmo tendo as providências para limpar tal recurso.

Figura 57 – Gráfico dos resultados da *view* na Tabela 7

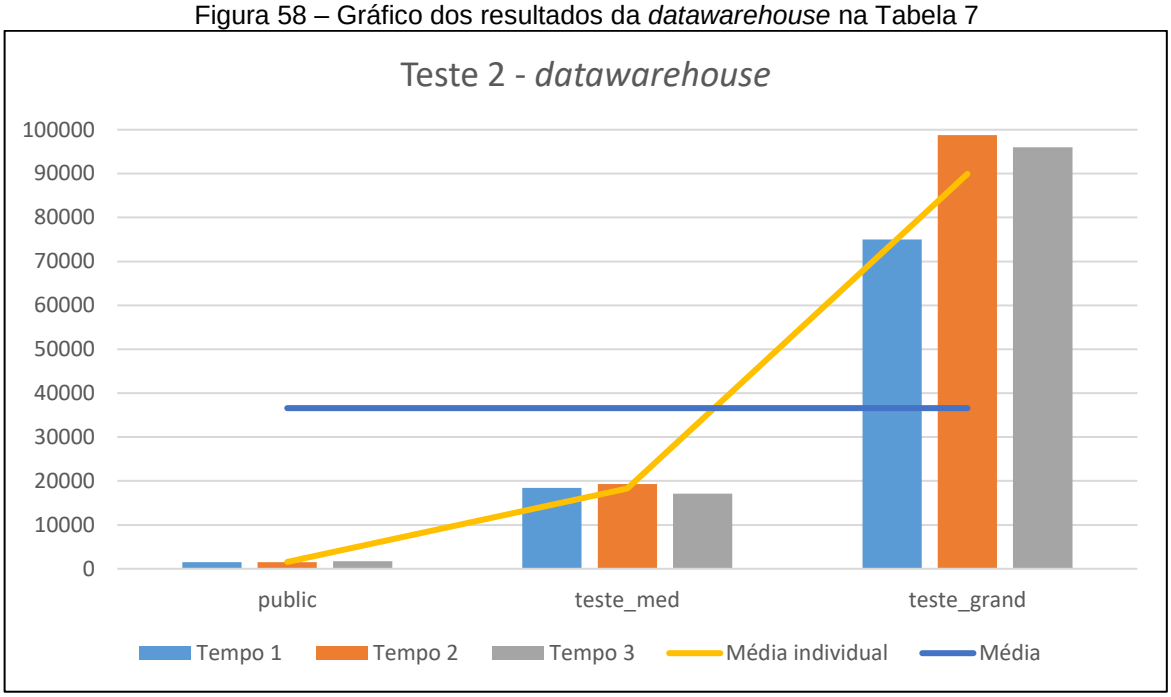


Fonte: Da Rosa (2016).

A ferramenta de pesquisa não permitirá realizar pesquisas sem nenhum filtro (cláusula *WHERE*[65]), sendo imprescindível ao menos um. O comportamento da Figura 57 é similar ao comportamento esperado ao gerar as pesquisas pelo portal. O número de linhas, apesar de igual entre as consultas, não influenciará tanto quando a diferença no volume de dados existentes no resultado da pesquisa desse teste. O comportamento das colunas, que são as amostras distintas de tempos executadas no mesmo esquema e em períodos diferentes, mostra que a diferença de tempo entre as amostras 1, 2 e 3 são mínimas.

[65] Comando SQL *WHERE* é utilizado para filtrar consultas. Geralmente, o conjunto de comandos sequentes a ele forma esse filtro sendo unidos por operadores como, por exemplo, *AND* e *OR*.

Figura 58 – Gráfico dos resultados da *datawarehouse* na Tabela 7



Fonte: Da Rosa (2006).

O terceiro teste utiliza o comando BETWEEN[66], pesquisando todos os nomes de organismos que contenham o número '1' ou a letra 'a', mas apenas nas linhas em que o endereço da fita 5 seja no mínimo 1/3 do seu maior valor e no máximo 2/3 do seu maior valor. Ou seja, suponha-se que o maior valor do endereço da fita 5 seja igual 9: a busca será realizada nas linhas em que esse endereço seja >=3 e <=6. A Tabela 8 mostra os tempos obtidos.

Tabela 8 – Tempos do 3º teste

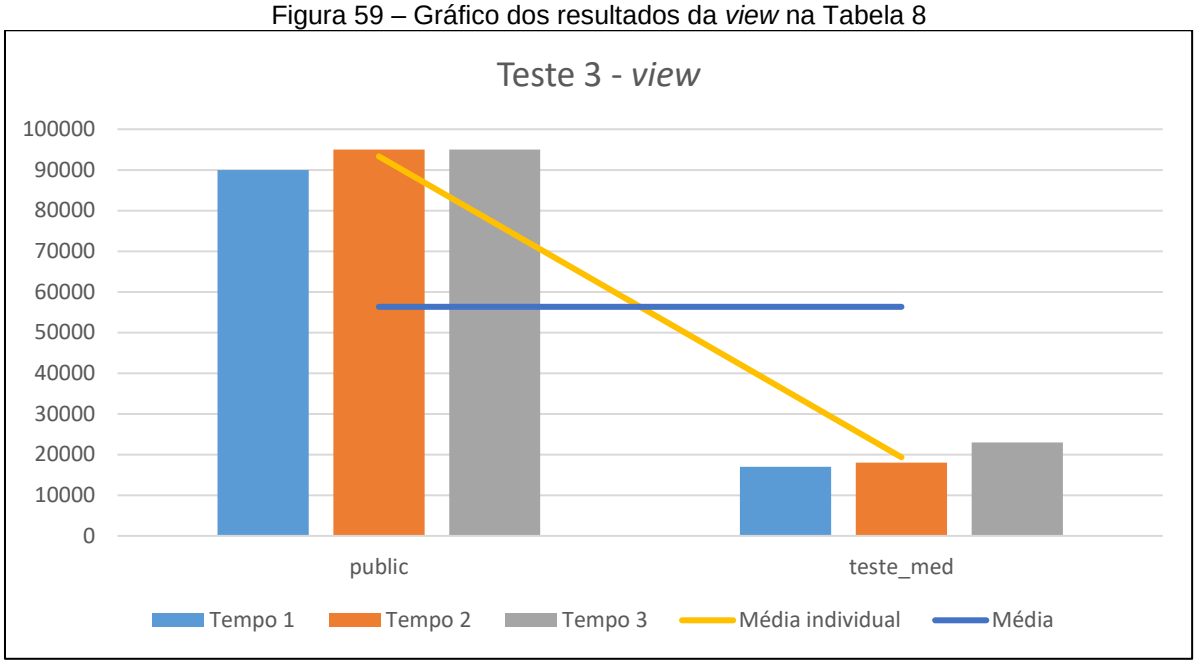
Esquemas	Origem	Tempo em milissegundos			
		Tempo 1	Tempo 2	Tempo 3	**Média individual**
public	view	90000	95000	95000	**93333,33**
	datawarehouse	2000	2300	2400	**2233,33**
teste_med	view	17000	18000	23000	**19333,33**
	datawarehouse	21000	19600	23100	**21233,33**
teste_grand	view	*out of memory*	*out of memory*	*out of memory*	**nulo**
	datawarehouse	36600	38800	60000	**45133,33**

Fonte: Da Rosa (2016).

[66] O comando *BETWEEN x AND y* filtra por valores >= x e valores <=y, ou seja, entre os valores x e y incluindo eles mesmos.

O tamanho resultante da consulta é relativa de um esquema para outro, pois os parâmetros no filtro depende do maior valor existente na coluna em análise. Novamente, a complexidade está na relação entre os dados e não somente a cargo do volume de dados das tabelas consultadas. O esquema *public*, Figura 59, mostrou dificuldade em gerir os relacionamentos e unir as informações, mesmo possuindo um número de linhas e volume inferior a *teste_med*.

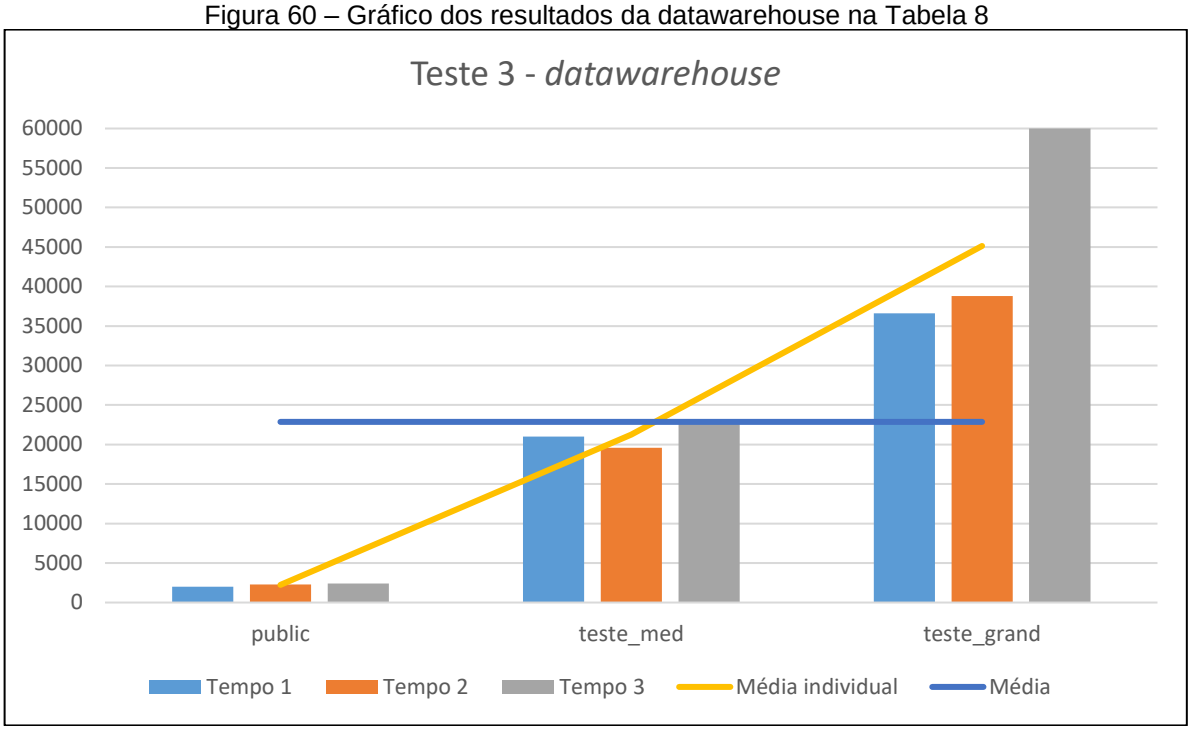
Figura 59 – Gráfico dos resultados da *view* na Tabela 8



Fonte: Da Rosa (2016).

Por sua vez, na Figura 60, o *datawarehouse* mostra que os mesmos dados tratados na *view* foram processados num espaço tempo melhor. O *teste_med* apresentou um desempenho similar de tempo na *datawarehouse* e *view*. Porém, a *teste_grand* na *view,* que apresentou problemas com o tamanho do resultado e na *datawarehouse,* possui tempo superior à *view no esquema public*.

Figura 60 – Gráfico dos resultados da datawarehouse na Tabela 8



Fonte: Da Rosa (2016).

A *view*, por outro lado, é mais simples de criar e gerenciar, mas peca quando o volume e a complexidade aumentam. A desvantagem é de não existir “fisicamente” no banco de dados como uma tabela. Quando requisitada, ela necessita executar uma consulta (que está dentro da sua estrutura lógica) que gera as suas linhas e colunas, ou seja, toda vez que ela está envolvida em uma consulta, haverá a necessidade dela atualizar-se ou criar-se (essa gestão é realizada pelo SGBD).

Quando houver simultâneas requisições à *view*, de diferentes sessões, cada sessão irá gerar a sua própria *view*, sobrecarregando o SGBD e o servidor. Isso requer uma *cache* com grande capacidade de armazenamento, incluindo memória RAM (atualmente o custo mais alto de um *hardware*). Como visto nos testes da *view* do esquema *teste_grand*, não foi possível obter nenhuma medida de tempo, pois a sessão ultrapassou a quantidade de recursos disponíveis.

Mudanças e manutenção na base como, por exemplo, inserções, atualizações, remoções ou limpeza do *cache*, forçam a *view* a realizar uma atualização dos seus dados em *cache*. Consequentemente, haverá um custo de tempo maior para a execução de uma pesquisa em que a *view* está envolvida.

Todas essas informações mostram que o uso de um *Data Warehouse* é superior a uma *view*. Uma vez que a DW é moldada à necessidade do sistema e os índices foram gerados, baseados nos testes, para atender cada necessidade,

buscando melhor desempenho, economizando espaço em disco e direcionando o PostgreSQL à usar melhor os índices.

5.1.2 Alteração do mecanismo de acesso ao banco de dados

Tornou-se inviável alterar ou substituir o *LP.Framework* na versão do portal IntergenicDB1.0 (I1), pois a quantidade de horas necessárias para corrigir os erros de código seriam inviáveis, não haveria garantida de que todos os recursos fossem testados. Além disso problemas atualmente enfrentados com o MySQL, a equipe do projeto optou pela migração do portal, gerando o projeto IntergenicDB2.0 (I2).

5.1.2.1 Mudança de *framework* e linguagem de programação

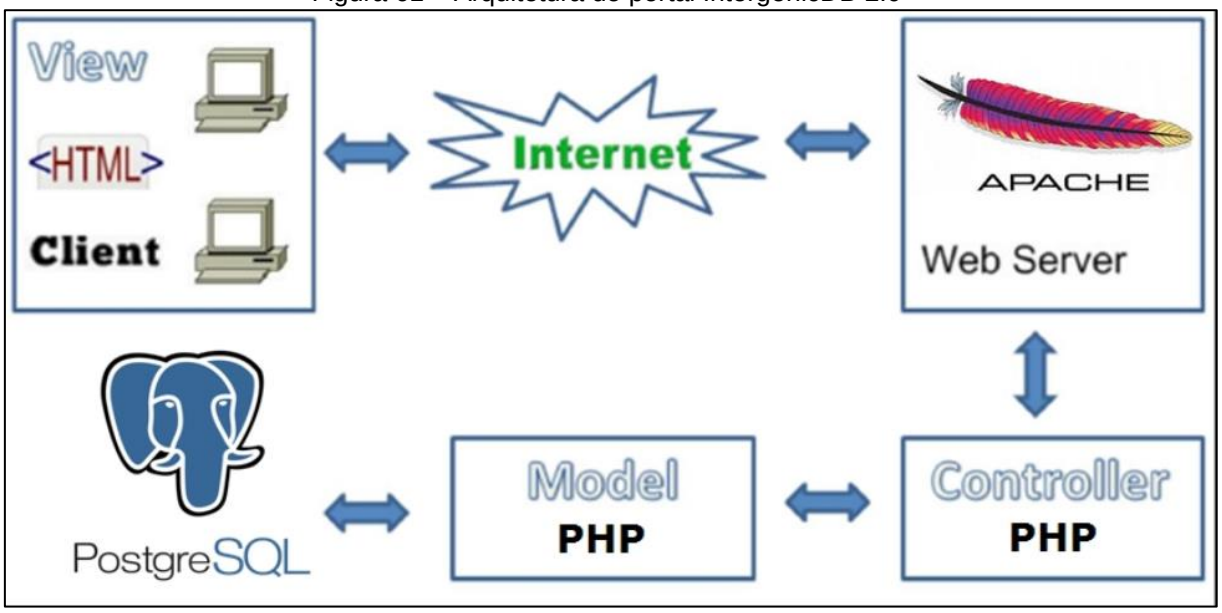
O portal I1 utilizava o padrão de arquitetura em MVC, que será mantido no novo portal I2. Esse padrão foi mantido para viabilizar o desenvolvimento da aplicação em paralelo, evitando conflitos de código fonte entre programadores, utilizando de programação orientada a objeto (POO)[67]. A alteração permite que a criação e manutenção da tela (parte gráfica, *View*), não interfira na regra de negócio (*Controller*), nem no acesso ao banco de dados e às consultas que serão realizadas (*Model*), pois estão divididas em camadas, seguindo a arquitetura.

A principal mudança foi a linguagem de programação, partindo da plataforma ASP .NET com C#, para PHP. Um dos principais motivos pela escolha da linguagem foi por ela ser *Open Source*[68], além de por ser uma linguagem de POO. O IIS[69], atual servidor *web*, não é compatível[70] com a linguagem PHP, sendo necessário migrar do

[67] A POO também era utilizada no portal I1 em linguagem *C#*.
[68] *Open Source*, ou código aberto (livre), não possui licença e nem é propriedade como a plataforma ASP .NET da Microsoft.
[69] *Internet Information Services*, servidor *web* da Microsoft.
[70] A incompatibilidade se deve ao fato dos administradores do servidor preferirem o servidor *web* Apache para rodar aplicações PHP, devido a políticas internas.

IIS para o Apache[71]. Com essas alterações, a nova arquitetura foi redesenha e apresentada na Figura 61.

Figura 61 – Arquitetura do portal IntergenicDB 2.0



Fonte: Andrade (2016).

O *framework* que será utilizado chama-se *CodeIgniter* (*CI*). Essa escolha se deu pelos seguintes motivos: compatibilidade (com o PHP, Apache e o PostgreSQL), por utilizar do padrão MVC e por ser *Open Source* (ANDRADE, 2016).

5.1.2.2 Testes

O quesito de testes verifica a viabilidade de ajustar ou substituir o *LP.Framework* sem prejudicar as demais *Controllers* e verificar se o número de modificações era viável. Tal objetivo não pôde ser alcançado e, para sanar esse e outros problemas, a equipe do projeto optou pela evolução do portal para a versão 2.0.

[71] *Apache* é um servidor *web Open Source*. As mesmas justificativas para a escolha do PHP aplicam-se.

5.1.3 Reconstrução da tela para o novo mecanismo de consulta

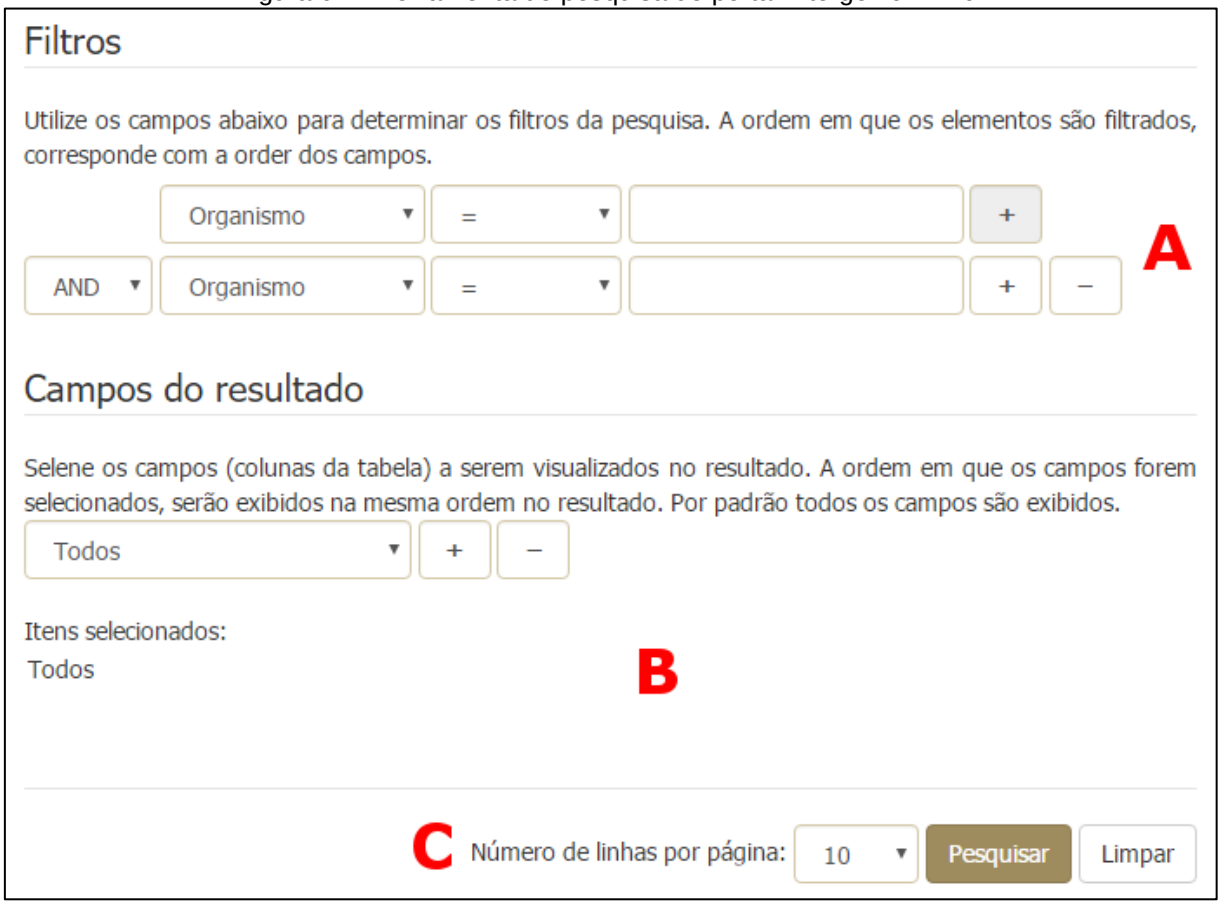
A implementação do *Data Warehouse*, tanto para I1 quanto para o I2, requer uma ferramenta de pesquisa não estática e que permita uma diversidade de filtros que atendam à necessidade do usuário do portal. O desenvolvimento dessa melhoria está descrito nessa sessão. Os rótulos dos botões, títulos, descrições e outros são provisórios, para atender à necessidade do autor durante o desenvolvimento e para viabilizar os testes, de modo que poderão ser alterados pelo proprietário.

5.1.3.1 Desenvolvimento e implementação de uma *query builder*

Uma *query builder*, ou construtor de consultas, permite que o usuário realize buscas mais refinadas através dos recursos disponibilizados. Assim, ao invés de realizar *'n'* consultas para localizar os dados desejados, os parâmetros de filtro podem ser unificados na mesma execução. Entretanto é preciso que o utilizador possua conhecimento prévio de operadores lógicos e noção básica de consultas em SQL.

A tela pesquisa desenvolvida (Figura 62), baseada no protótipo (Figura 35 e 36), seguiu o novo layout da versão 2.0 e os padrões de CSS definidos por Andrade (2016). A tela está partilhada em três: o filtro da pesquisa (Figura 62 A), os campos (colunas) que serão exibidos na tela de resultado (Figura 62 B) e a paginação do resultado (Figura 62 C).

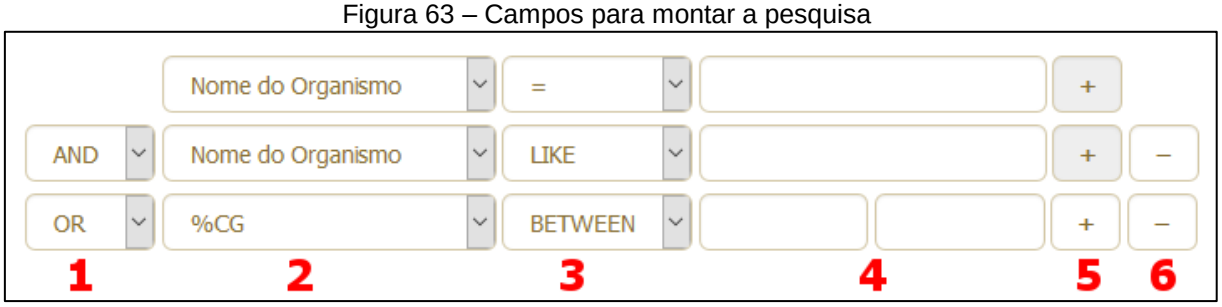
Figura 62 – Ferramenta de pesquisa do portal IntergenicDB2.0



Fonte: Da Rosa (2016).

O primeiro item, Figura 62 A, é a ferramenta gráfica que auxiliará o usuário na montagem do filtro da consulta. É obrigatório que, para realizar a pesquisa, pelo menos um item deva existir e estar devidamente preenchido para filtrar os dados. A primeira coluna, Figura 63, não é ativa para a primeira linha, trata-se dos operadores de união (conjuntivos) *AND* (operador e) e *OR* (operador ou) da cláusula *WHERE*.

Figura 63 – Campos para montar a pesquisa



Fonte: Da Rosa (2016).

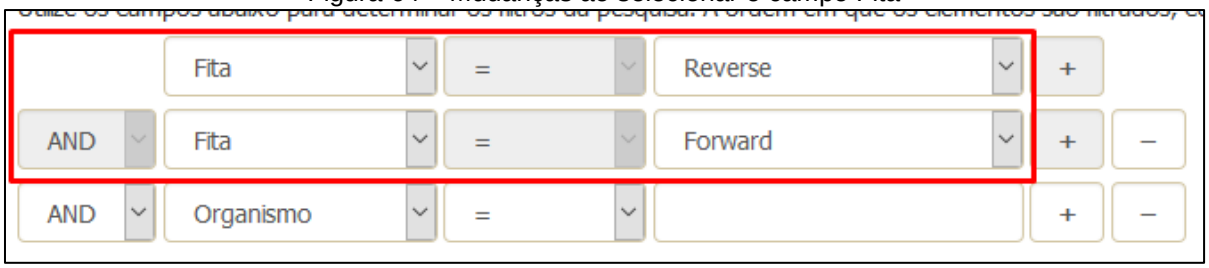
A segunda coluna refere-se às colunas de uma determinada tabela. Esse campo não necessariamente terá o mesmo nome da coluna e está graficamente

representada para facilitar a compreensão. Conforme já referido, o administrador do portal poderá redefinir os rótulos. A coluna de número três representa os operadores lógicos que serão selecionados pelo usuário, mas nem todo o operador visível na lista é compatível com o campo a sua esquerda (coluna 2) como, por exemplo:

a) os operadores >, >=, <, <= e *BETWEEN*, somente devem ser utilizados para campos do tipo numérico;
b) operador *LIKE* somente é utilizado para campos do tipo texto e serve para procurar um trecho dentro de uma palavra ou sentença;

A coluna 4 é, para a maioria, um campo livre para a digitação. exceto quando selecionado na coluna 2 o campo que representa o sentido da fita de uma região intergênica. O usuário pode selecionar entre *Forward* – em frente que representa o sentido 5'→ 3' – ou *Reverse* – o inverso que representa o sentido 3'→ 5'. Quando selecionado o item Fita na coluna 2, uma modificação ocorrerá na tela, como ilustrado na Figura 64. As colunas 1 e 3 são bloqueadas, por se tratar de um campo do tipo binário, e a coluna 4 será um campo de seleção.

Figura 64 – Mudanças ao selecionar o campo Fita



Fonte: Da Rosa (2016).

O operador LIKE requer que o usuário conheça e utilize os curingas *'%'* e *'_'*. O caractere *'%'* representa uma sequência de um ou mais caracteres, enquanto o *'_'* representa um único caractere. O mecanismo está tratando a consulta, com LIKE, *case sensitive*[72]. Alguns exemplos de como utilizá-los:

a) %a% - retorna as linhas que contenham a letra "a";
b) %A - retorna as linhas que terminam com a letra "A";
c) bacteria% - retorna as linhas que começam com a sequência "bacteria";

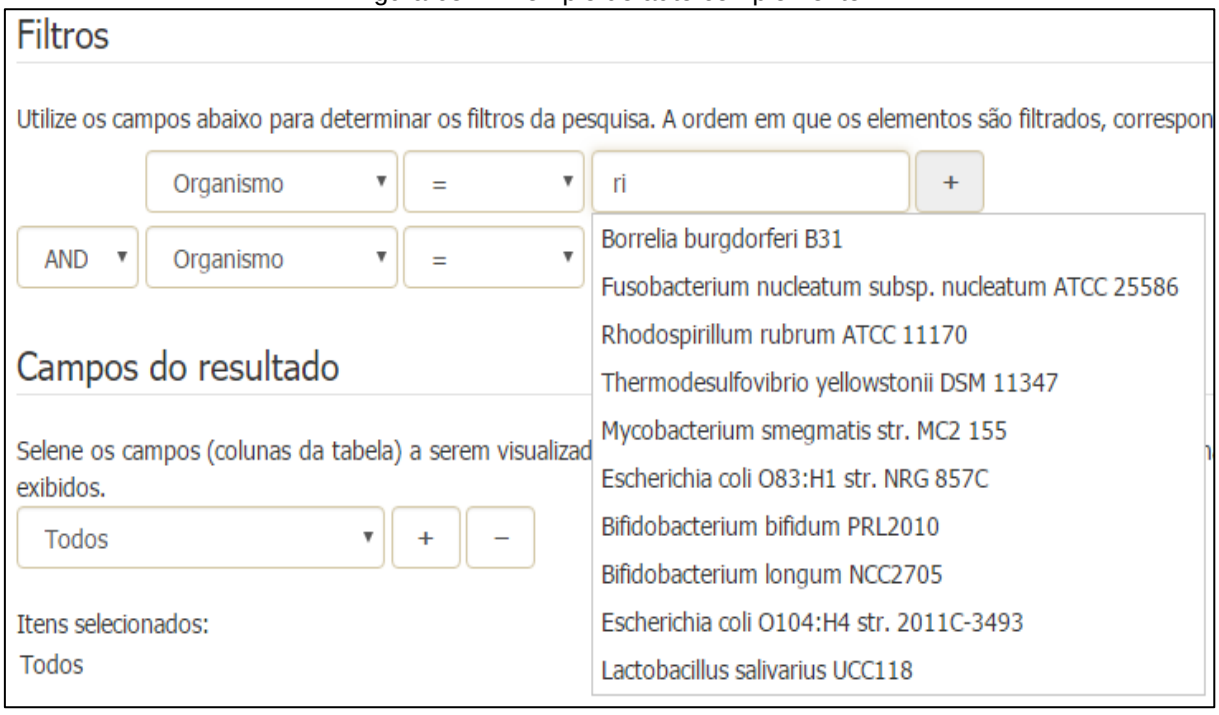
[72] Oposto do *case insensitive*.

d) %a%C% - retorna as linhas que contenham a letra “a”, seguidas em algum momento da sequência de caracteres que contenham ou terminem com a letra “C”;
e) B_cteria – retorna as linhas que começam com a letra “B”, seguidos obrigatoriamente de um caractere qualquer e terminando com a cadeia “cteria”;
f) a_% - retorna linhas que começam com a letra “a”, seguidos obrigatoriamente de um caractere e terminando com qualquer sequência de ou nada;

Outros exemplos e instruções de uso estão disponível no manual criado e incluído no portal IntergenicDB 2.0 no menu de Ajuda.

A quarta coluna ainda conta com o recurso de auto complemento, que é ativado somente para os campos de texto. O número máximo de sugestões na lista é de 10 linhas. Na Figura 65, vê-se um exemplo ao digitar *“ri”* (dois dígitos), exibindo a lista com sugestões, ou seja, itens da coluna de “Nome do Organismo” que possuem os caracteres *“ri”.* Esse recurso é *case insensitive*.

Figura 65 – Exemplo do auto complemento

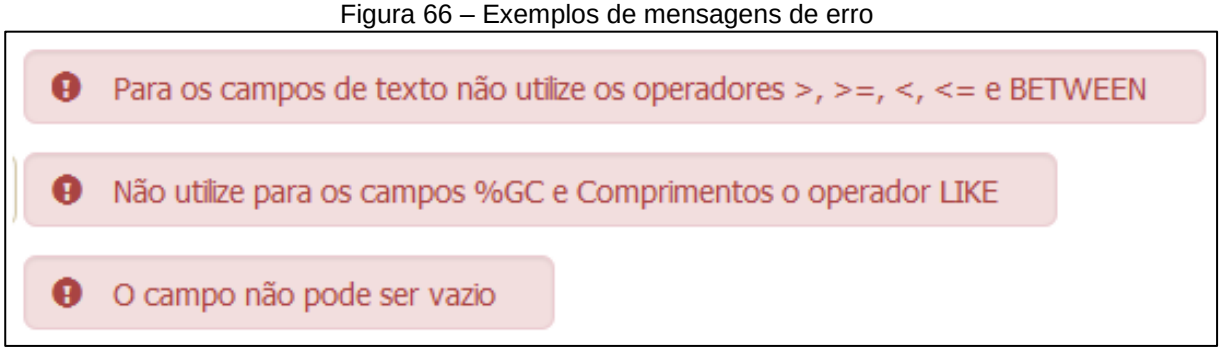


Fonte: Da Rosa (2016).

Caso seja (na coluna 4 da Figura 63) deixado um campo em branco ou digitado um valor incompatível com o tipo do campo ou com o tipo do operador, aparecerá uma

mensagem de erro ao lado direito da coluna 6 – podendo haver quebras de linhas dependendo da resolução (tamanho) da janela do browser – da linha será exibida. Na Figura 66, segue alguns exemplos.

Figura 66 – Exemplos de mensagens de erro

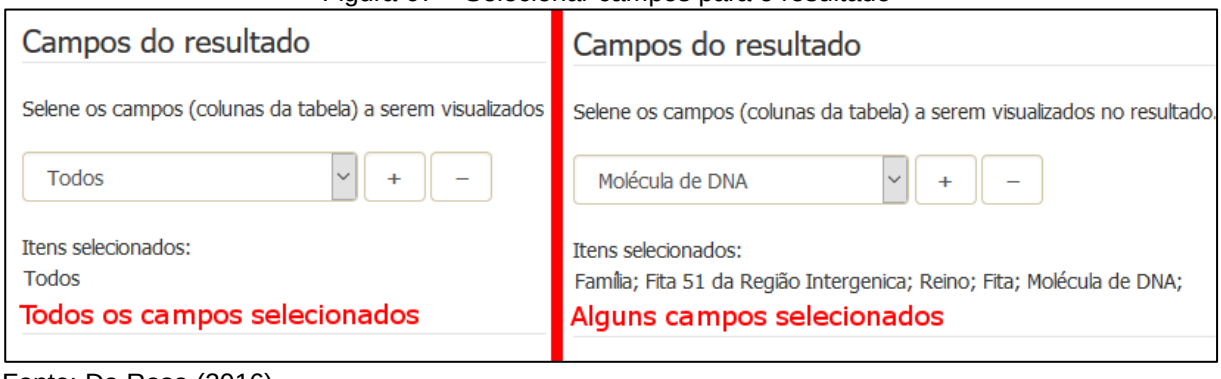


Fonte: Da Rosa (2016).

As colunas 5 e 6 são para adicionar novas linhas ao filtro ou removê-las. A inclusão e a remoção são feitas usando recursos de *JavaScript*, o que permite a inserção de até 10 linhas de filtros, no máximo, e não é possível remover a primeira linha.

A Figura 62 B permite a seleção das colunas que o usuário deseja ver no resultado. Por padrão, todos os campos estão selecionados. Caso não seja necessário algum deles, pode-se utilizar os botões de adição e remoção do item selecionado. Não é possível deixar sem nenhum item selecionado. A Figura 67 mostra a diferença da tela quando selecionados alguns itens ou todos os itens.

Figura 67 – Selecionar campos para o resultado



Fonte: Da Rosa (2016).

O item C da Figura 62 conterá a opção para determinar quantas linhas cada paginação. O recurso permite a escolha de um número pelo usuário entre 10, 25, 50 e 100. Esses números representam a quantidade de linhas que serão exibidas por

página na tela de resultado. Por exemplo, suponha-se que o total de linhas da consulta seja 1000 e o número de linhas escolhido seja igual a 50; na tela de resultado, serão apresentadas 20 páginas com 50 linhas cada. Mas, antes de permitir ao usuário definir tal parâmetro, ou iniciar uma pesquisa clicando no botão “Pesquisar”, ou limpar a tela retornando ao estado inicial, uma verificação de segurança, chamada de *reCAPTCHA* (Figura 67), deverá ser realizada.

Figura 68 – Caixa de checagem de segurança (*reCAPTCHA*)

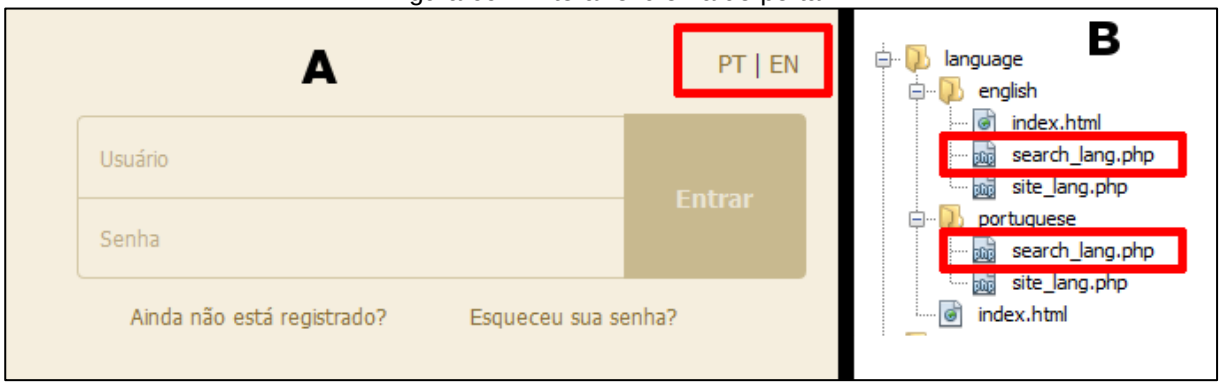
Fonte: Da Rosa (2016).

O *reCAPTCHA* é uma API de proteção, fornecida gratuitamente pela Google, para aumentar a segurança de portais contra *spam* e outros tipos de recursos abusivos que aproveitam determinadas falhas de segurança. No caso do portal, o *JavaScript*[73] é um ponto fraco podendo ser utilizado por ferramentas que farão diversas requisições (pesquisas) aleatórias buscando parar o serviço ou os servidores. Para exibir os comandos da tela, primeiro deve ser feita a verificação marcando a caixa de checagem (*checkbox*) *“I’m not a robot”*, ou seja, “Eu não sou um robô”, referência aos diversos tipos de *spam* e ferramentas maliciosas da *web*.

O *framework CodeIgniter* possui o recurso de internacionalização, ou seja, modificar o idioma do portal entre português e inglês. O idioma padrão do portal será definido conforme as configurações locais do computador e do browser. Sendo possível alterar, caso necessário, através da Figura 69 A. Para cada idioma, há uma pasta e seus respectivos arquivos (Figura 69 B).

[73] O *JavaScript* é uma linguem utilizada em páginas da Internet para a execução de determinadas funções na máquina do utilizador, e não no servidor. Isso permite uma brecha de segurança a ser explorada, mas pode ser protegida por recursos como, por exemplo, *reCAPTCHA*.

Figura 69 – Alterar o idioma do portal



Fonte: Da Rosa (2016).

Para a alteração dos rótulos, deve-se seguir as instruções na forma de comentários da linguagem PHP, existente dentro de cada arquivo. Como mostrado na Figura 70, os textos entre colchetes não devem ser alterados. Somente os textos após o sinal de igual (=) podem ser modificados conforme a necessidade, e devem estar entre aspas simples (‘texto de exemplo’) ou aspas duplas (“texto de exemplo”).

Figura 70 – Exemplo de instrução para alterar os rótulos do portal

```
/*
 * Rótulos dos check box
 * $lang['dont change']['dont_change_tableName_collumnName'] = 'Label change';
 * $lang['não altere'] ['não_altere_tableName_collumnName']  = 'Rotuló altere';
 */
  $lang['checkbox']   ['family_name']                        = 'texto a exibir';
```

Fonte: Da Rosa (2016).

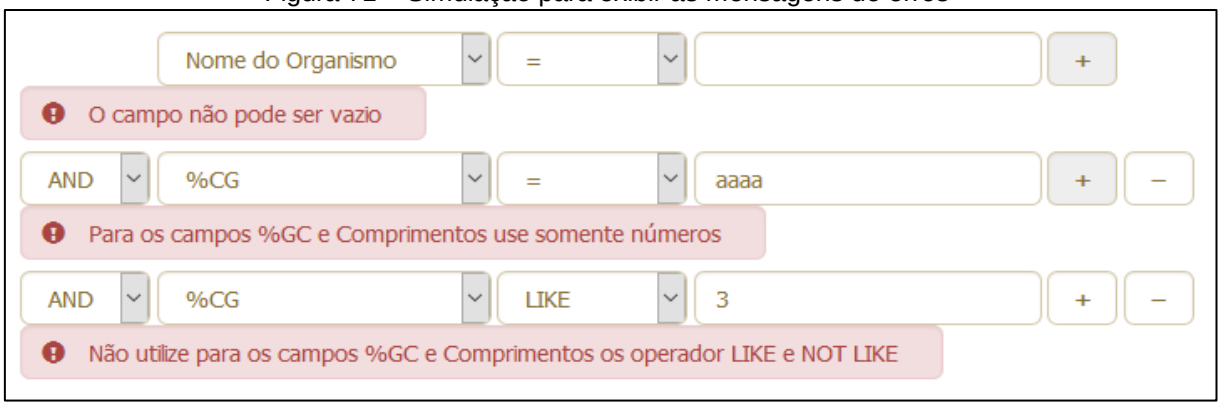
5.1.3.2 Testes

O primeiro critério de avalição será testado em conjunto com a tela de resultados na sessão 5.1.4. Empregando os mesmos testes para validar as informações recebidas pela tela de resultado, também, é possível validar os dados enviados pela tela de pesquisa. Por essa razão, ambos os testes serão realizados na próxima sessão.

A segunda avaliação é quanto aos recursos de adicionar linhas e remove-las. Ao adicionar linhas e tentar realizar a pesquisa, preenchendo os campos de maneira

incompatível entre o tipo do campo e o operador, as mensagens de alerta são exibidas de acordo com o erro do usuário (Figura 71).

Figura 71 – Simulação para exibir as mensagens de erros

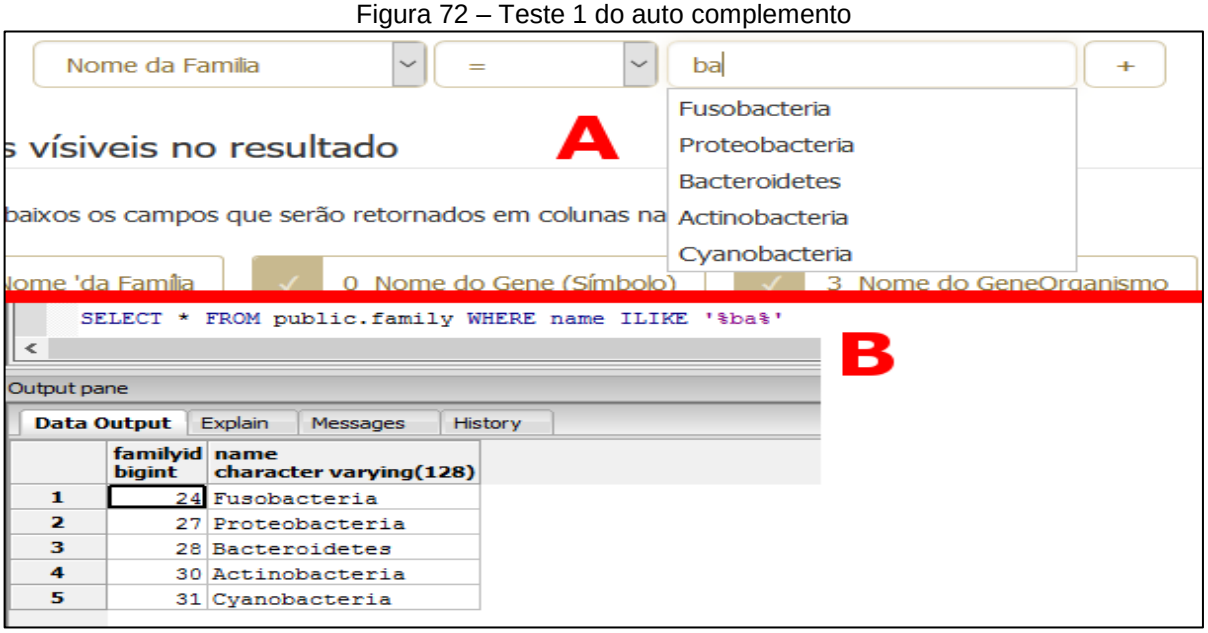


Fonte: Da Rosa (2016).

No momento em que removem-se as duas últimas linhas e estando o campo da primeira linha devidamente utilizado, a mensagem permanece até que seja clicado no botão *"Pesquisar".* A validação é executada novamente, a mensagem de erro é eliminada, por atender os requisitos, e a pesquisa é executada. A integridade dos itens selecionados, ao remover ou inserir linhas, também será testado na sessão 5.1.4.

O auto complemento, como mencionado, somente é ativado para campos do tipo texto, a partir do segundo digito. A Figura 72 mostra, no quadro B, uma consulta semelhante à executada pelo recurso de auto complemento. Em ambos os casos (A e B), temos os mesmos resultados: 5 itens.

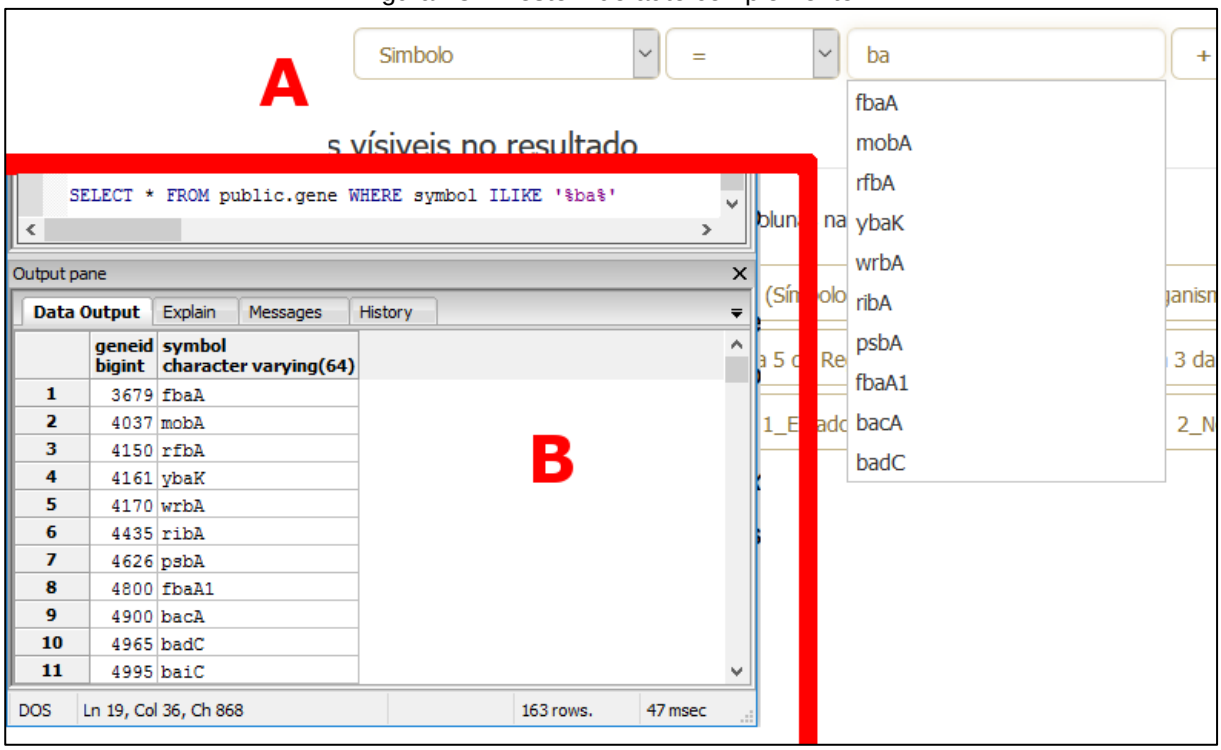
Figura 72 – Teste 1 do auto complemento



Fonte: Da Rosa (2016).

Nesta outra simulação, Figura 73, os caracteres *“ba”* são utilizados no campo “símbolo dos genes”. Ao acrescentar os dígitos no campo, o recurso mostra os dez primeiros símbolos encontrados, conforme definido pelo usuário. Na consulta, no pgAdmin3, o resultado chega a 163 registros e os seus 10 primeiros correspondem ao exibido em tela. Não é função do recurso fazer uma consulta com tantas linhas. Seu objetivo é facilitar a montagem da consulta.

Figura 73 – Teste 1 do auto complemento



Fonte: Da Rosa (2016).

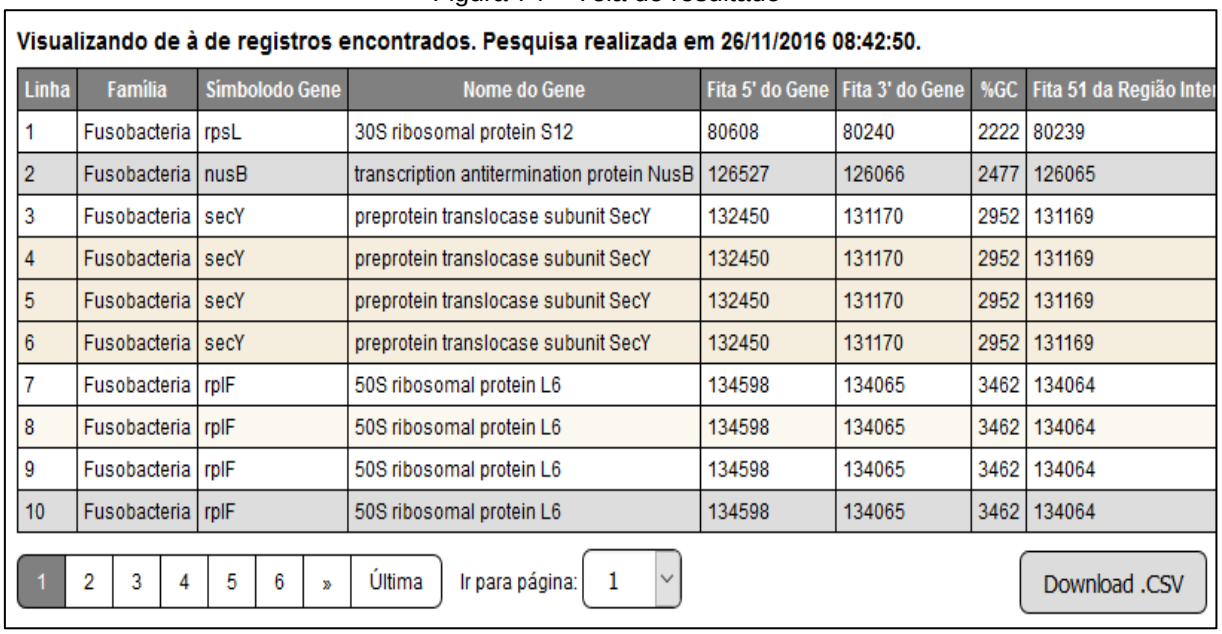
5.1.4 Remodelagem da tela de resultados da pesquisa

5.1.4.1 Implementação

A nova tela de resultado possui um *layout* simples como o proposto no protótipo da Figura 37, fundo branco com letras, traços e linhas pretas e tons de cinza. O objetivo é permitir uma visualização simples e clara dos dados. Além disso, possibilita

a interação com o usuário através dos controles de navegação e seleção de linhas. Na Figura 74 temos um exemplo da tela de resultado.

Figura 74 – Tela de resultado



Visualizando de à de registros encontrados. Pesquisa realizada em 26/11/2016 08:42:50.

Linha	Família	Simbolodo Gene	Nome do Gene	Fita 5' do Gene	Fita 3' do Gene	%GC	Fita 51 da Região Inte
1	Fusobacteria	rpsL	30S ribosomal protein S12	80608	80240	2222	80239
2	Fusobacteria	nusB	transcription antitermination protein NusB	126527	126066	2477	126065
3	Fusobacteria	secY	preprotein translocase subunit SecY	132450	131170	2952	131169
4	Fusobacteria	secY	preprotein translocase subunit SecY	132450	131170	2952	131169
5	Fusobacteria	secY	preprotein translocase subunit SecY	132450	131170	2952	131169
6	Fusobacteria	secY	preprotein translocase subunit SecY	132450	131170	2952	131169
7	Fusobacteria	rplF	50S ribosomal protein L6	134598	134065	3462	134064
8	Fusobacteria	rplF	50S ribosomal protein L6	134598	134065	3462	134064
9	Fusobacteria	rplF	50S ribosomal protein L6	134598	134065	3462	134064
10	Fusobacteria	rplF	50S ribosomal protein L6	134598	134065	3462	134064

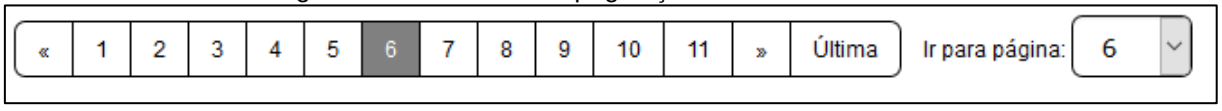
1 2 3 4 5 6 » Última Ir para página: 1 Download .CSV

Fonte: Da Rosa (2016).

Além desses controles, ao passar o mouse sobre uma linha (Figura 74, linha 8) a cor de fundo será alterada para destacá-las em relação às demais. Caso o usuário deseje manter a marcação (selecionar) uma ou mais linhas, pode ser feito com um clique do mouse sobre tal. Um exemplo pode ser visto na Figura 74, linhas 4, 5 e 6.

O botão de *"Download .CSV"* permitirá que o usuário realize o download de todos os dados da consulta em um único arquivo texto, que será definido na sessão 5.1.5. Por sua vez os controles de paginação, Figura 75, fornecem recursos de navegação dentro tela de resultado. A quantidade de páginas será relativa, de acordo com o número de linhas solicitado pelo usuário na tela de pesquisa e a quantidade total de linhas da consulta.

Figura 75 – Controles de paginação da tela de resultado



Fonte: Da Rosa (2016).

5.1.4.2 Testes

Para testar se a consulta montada na tela de pesquisa é a mesma exibida na tela de resultado, será modificado o código – modificação temporária para realizar os testes – para que o código retorne na tela as informações selecionadas pelo usuário na tela de pesquisa e a *string* de consulta gerada. Essa *string* irá ser executada no SGBD pgAdmin3 e deverá que ter como resultado as mesmas linhas que estão em tela.

A primeira simulação procura por nomes de organismos que contenham o texto *“Chloroflexus”* com todas as colunas disponíveis. Na Figura 76, a matriz – também conhecida por *array multidimensional* ou vetor multidimensional – reúne os parâmetros informados definidos pelo usuário na tela de pesquisa e impressos na tela de resultado. Os itens da Figura 62 B são armazenados no índice “*select”* da matriz.

Figura 76 – Informações vindas da tela de pesquisa

```
post data

Array
(
    [tables] => Array
        (
            [organism] => o
            [family] => f
            [gene] => g
            [geneorganism] => go
            [intergenicregion] => ir
            [kingdom] => k
            [role] => r
        )

    [wh] => Array
        (
            [0] => Array
                (
                    [0] => AND
                    [1] => o.name =
                    [2] => Chloroflexus aurantiacus J-10-fl
                )

        )

    [select] => Array
        (
            [0] => f.name AS family_name
            [1] => g.symbol AS gene_symbol
            [2] => go.name AS geneorganism_name
            [3] => go.tape5address AS geneorganism_tape5address
            [4] => go.tape3address AS geneorganism_tape3address
            [5] => go.gcpercentage AS geneorganism_gcpercentage
            [6] => ir.tape5address AS intergenicregion_tape5address
            [7] => ir.tape3address AS intergenicregion_tape3address
            [8] => ir.sequence AS intergenicregion_sequence
            [9] => ir.length AS intergenicregion_length
```

Fonte: Da Rosa (2016).

Cada linha de filtro, Figura 62 A, é armazenada no índice *“wh”* da matriz. Cada sub índice ([0], [1], [2], [...], [10]) corresponde, respectivamente, a uma linha da tela de pesquisa. Já, o índice *“tables*”, que será utilizado para montar a união entre as tabelas (comandos *JOIN*), precisa de um tratamento para dar um melhor desempenho à consulta, não requer que todas as tabelas estejam em seus sub níveis. O mecanismo de busca, a *Controller Pesqdw.php*, utiliza as funções *index()* – que inicializa a *Controller* – e *setDataForm()* – auxiliando a função inicial nessa estruturação dos dados e seleção das tabelas envolvidas na união. Essas funções estão representadas na Figura 77 e destacam a linha que alimenta o vetor *“tables”*.

Figura 77 – Código fonte do mecanismo de consulta – parte 1 (Pesqdw.php)

```
59
60      private function setDataForm($num) {
61          $tmp = explode('.',
62                  str_replace("_", ".", (string) $this->input->post(
63                          $this->data['index'][1] . $num)));
64          $abr = $this->tableAbbreviation($tmp[0]);
65          $cp = $abr . "." . $tmp[1];
66          $cond = (string) $this->input->post($this->data['index'][2] . $num);
67          $wh_op = (string) $this->input->post($this->data['index'][0] . $num);
68          $this->dtsearch['tables'][$tmp[0]] = $abr;
69          if ($cp == 'ir.status') {...4 linhas } else {...4 linhas }
76      }
        function index($id_session = NULL) {
78          /*...2 linhas */
80          $config['uri_segment'] = 4;
81
82          if($this->input->post()){
83              $id_session = time(); $fim = false; $i = 0;
84              do {...5 linhas } while (!$fim);
89              $tmp = array();
90              if ($this->input->post(
91                      $this->data['index'][5]) != "all"){...5 linhas }
96              else{...5 linhas }
                $this->data['tablelabel'] = $tmp; $sztmp = count($tmp);
102             foreach ($tmp as $key => $value) {
103                 $tmp = explode('_', $value);
104                 $this->dtsearch['select'][$key] = $this->tableAbbreviation(
105                         $tmp[0]).".".$tmp[1]." AS ".$value;
106                 $this->dtsearch['tables'][$tmp[0]] = $this->tableAbbreviation(
107                         $tmp[0]);
108             }
109             $this->dtsearch['tz'] = $this->input->post('tz');
110             $this->data["dtsearch"] = $this->dtsearch;
```

Fonte: Da Rosa (2016).

As últimas etapas da função *index()*, antes de imprimir em tela o vetor para analisar o teste e a tabela de resultados, seriam:

a) contar o número de linhas totais que a pesquisa retornaria, para assim poder configurar a paginação. Realizada pela função *Datawarehouse_model-> get_dw_count()*, item 1 da Figura 78;

b) realizar a consulta ao banco de dados utilizando os parâmetros para limitar o número de linhas definido pelo usuário. Executado pela função *Datawarehouse_model-> get_dw_where()*, item 2 da Figura 78;

c) retornar da *Model*, *Datawarehouse_model, a última consulta executa.* Função *Datawarehouse_model-> get_dw_dump()*, item 3 da Figura 78;

d) armazenar na sessão os dados para permitir a navegação (paginação) e a utilização de mais telas de resultado com diferentes consultas montadas na tela de pesquisa. Comandos do item 4 da Figura 78.

Figura 78 – Código fonte do mecanismo de consulta – parte 2 (Pesqdw.php)

```
108             }
109             $this->dtsearch['tz'] = $this->input->post('tz');
110             $this->data["dtsearch"] = $this->dtsearch;
111             $config["per_page"] = $this->input->post($this->data['index'][6]);
112             $config['base_url'] = site_url('pesqdw/index/'.$id_session);
113             $config['total_rows'] = $this->dw->get_dw_count(
114 1                   $this->dtsearch['select'], $this->dtsearch['tables'],
115                     $this->dtsearch['wh']);
116             $this->data['linhas'] = $this->dw->get_dw_where(
117                     $this->dtsearch['select'], $this->dtsearch['tables'],
118 2               $this->dtsearch['wh'], $config['per_page'],
119                     (int) $this->uri->segment($config['uri_segment']));
120             $this->data['dump'] = $this->dw->get_dw_dump(
121 3                   $this->dtsearch['select'], $this->dtsearch['tables'],
122                     $this->dtsearch['wh']);
123             $dtsearch_base64 = base64_encode(json_encode($this->dtsearch));
124             $this->data['dtsearch'] =
125                     '<input type="hidden" name="sess" id="sess" value="'
126                 .$id_session.'"/>'
127                     . '<input type="hidden" name="dtv" id="dtv" value="'
128                 .$dtsearch_base64.'"/>';
129             $_SESSION[$id_session.'_per_page'] = $config['per_page'];
130 4           $_SESSION[$id_session.'_total_rows'] =  $config['total_rows'];
131             $_SESSION[$id_session.'_dump'] = $dtsearch_base64;
132             $_SESSION[$id_session.'_vcampos'] = $this->data['tablelabel'];
133         }else{
134             $this->data['tablelabel'] = $_SESSION[$id_session.'_vcampos'];
135             $config['base_url'] = site_url('pesqdw/index/'.$id_session);
```

Fonte: Da Rosa (2016).

A *string* de consulta gerada é impressa na tela conforme o exemplo da Figura 79. A função *get_dw_dump()* somente solicita a consulta em SQL montada e executada pelas demais funções da *Model* (*Datawarehouse_model.php)* e da Controller (*Pesqdw.php*). Não necessita de acesso ao banco de dados.

Figura 79 – Consulta criada pelo mecanismo

query sql

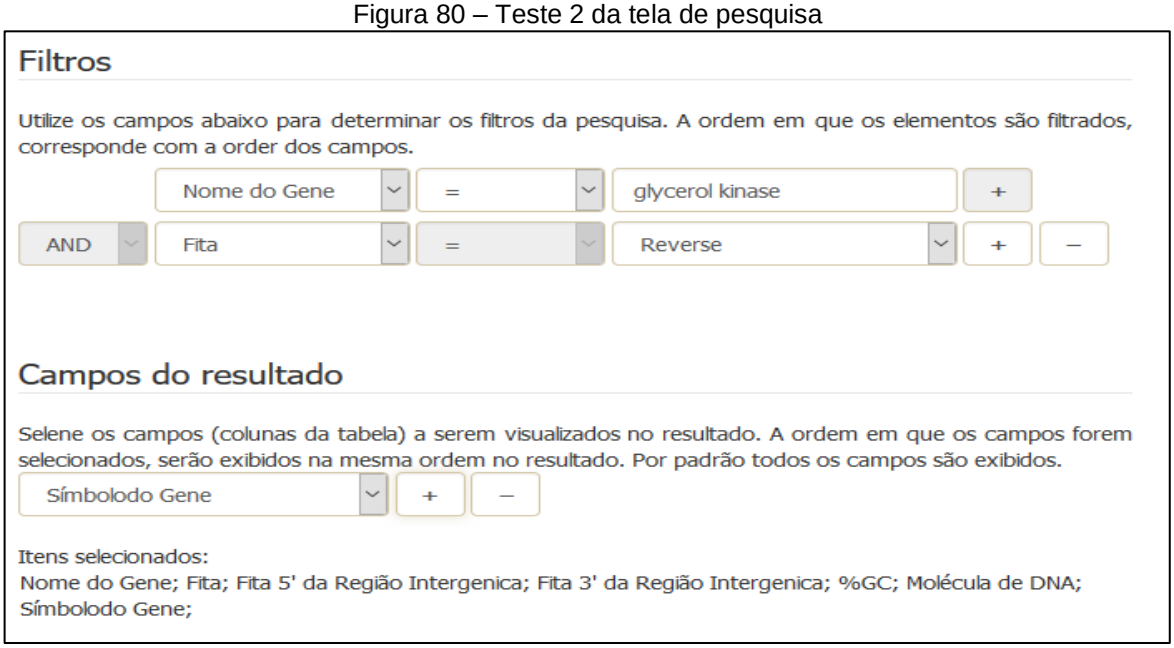
```
SELECT "f"."name" AS "family_name",
 "g"."symbol" AS "gene_symbol",
 "go"."name" AS "geneorganism_name",
 "go"."tape5address" AS "geneorganism_tape5address",
 "go"."tape3address" AS "geneorganism_tape3address",
 "go"."gcpercentage" AS "geneorganism_gcpercentage",
 "ir"."tape5address" AS "intergenicregion_tape5address",
 "ir"."tape3address" AS "intergenicregion_tape3address",
 "ir"."sequence" AS "intergenicregion_sequence",
 "ir"."length" AS "intergenicregion_length",
 "ir"."status" AS "intergenicregion_status",
 "k"."name" AS "kingdom_name",
 "o"."name" AS "organism_name",
 "o"."dnamolecule" AS "organism_dnamolecule",
 "r"."name" AS "role_name"
FROM "datawarehouse" AS "dw"
INNER JOIN "organism" AS "o" ON "dw"."organismid" = "o"."organismid"
INNER JOIN "family" AS "f" ON "dw"."familyid" = "f"."familyid"
INNER JOIN "gene" AS "g" ON "dw"."geneid" = "g"."geneid"
INNER JOIN "geneorganism" AS "go" ON "dw"."geneorganismid" = "go"."geneorganismid"
INNER JOIN "intergenicregion" AS "ir" ON "dw"."intergenicregionid" = "ir"."intergenicregionid"
INNER JOIN "kingdom" AS "k" ON "dw"."kingdomid" = "k"."kingdomid"
INNER JOIN "geneorganismrole" AS "gor" ON "dw"."geneorganismid" = "gor"."geneorganismid"
INNER JOIN "role" AS "r" ON "dw"."roleid" = "r"."roleid"
WHERE "o"."name" LIKE '%Chloroflexus%'
```

Fonte: Da Rosa (2016).

A Figura 76 e Figura 79 mostram que os parâmetros de dados utilizados pelo usuário na tela de pesquisa, foram atendidos na primeira simulação, sendo que, ao executar a consulta (Figura 79) no programa pgAdmin3, os dados presentes na tela de resultado e no programa são os mesmos.

O segundo teste está ilustrado na Figura 80, através do qual deseja-se consultar por genes que tenham o nome igual à *"glycerol kinase"*, e cuja fita esteja no sentido contrário. Foram selecionados alguns campos para serem visíveis na tela de resultado.

Figura 80 – Teste 2 da tela de pesquisa



Fonte: Da Rosa (2016).

Ao executar a pesquisa, as informações são estruturadas na matriz conforme o programado (Figura 81). O resultado é exibido em tela e as informações na tabela de resultado conferem com a execução da *string* de consulta (Figura 82) no pgAdmin3.

Figura 81 – Matriz de parâmetros da consulta gerada a partir do teste 2

```
Array
(
    [tables] => Array
        (
            [geneorganism] => go
            [intergenicregion] => ir
            [organism] => o
            [gene] => g
        )

    [wh] => Array
        (
            [0] => Array
                (
                    [0] => AND
                    [1] => go.name =
                    [2] => glycerol kinase
                )

            [1] => Array
                (
                    [0] => AND
                    [1] => ir.status =
                    [2] => Reverse
                )

        )

    [select] => Array
        (
            [0] => go.name AS geneorganism_name
            [1] => ir.status AS intergenicregion_status
            [2] => ir.tape5address AS intergenicregion_tape5address
            [3] => ir.tape3address AS intergenicregion_tape3address
            [4] => go.gcpercentage AS geneorganism_gcpercentage
            [5] => o.dnamolecule AS organism_dnamolecule
            [6] => g.symbol AS gene_symbol
        )

    [tz] => 120
)
```

Fonte: Da Rosa (2016).

Figura 82 – Consulta gerada pelo mecanismo de pesquisa para o teste 2

```
SELECT "go"."name" AS "geneorganism_name",
"ir"."status" AS "intergenicregion_status",
"ir"."tape5address" AS "intergenicregion_tape5address",
"ir"."tape3address" AS "intergenicregion_tape3address",
"go"."gcpercentage" AS "geneorganism_gcpercentage",
"o"."dnamolecule" AS "organism_dnamolecule",
"g"."symbol" AS "gene_symbol"
FROM "datawarehouse" AS "dw"
INNER JOIN "geneorganism" AS "go" ON "dw"."geneorganismid" = "go"."geneorganismid"
INNER JOIN "intergenicregion" AS "ir" ON "dw"."intergenicregionid" = "ir"."intergenicregionid"
INNER JOIN "organism" AS "o" ON "dw"."organismid" = "o"."organismid"
INNER JOIN "gene" AS "g" ON "dw"."geneid" = "g"."geneid"
WHERE "go"."name" = 'glycerol kinase' AND "ir"."status" = 'R'"
```

Fonte: Da Rosa (2016).

A tela de consulta está atendendo ao que foi proposto. Os parâmetros para filtrar a pesquisa estão sendo enviados para o mecanismo de consulta e os dados estão sendo validados antes de serem transmitidos, evitando assim diversas requisições invalidas ao mecanismo de busca.

A tela de resultado recebe os dados corretamente, executa um tratamento para entregar um vetor do qual será originado o resultado da pesquisa. Além disso, o mecanismo de busca é responsável por configurar a paginação e imprimir os dados em tela. A formatação do resultado está conforme aprovado pelo usuário, tendo um *layout* simples e separado do ambiente em que realizamos a consulta. Dessa forma, é possível ter diversos resultados, em abas distintas, criadas a partir de diferentes consultas realizadas.

5.1.5 Formato de arquivo para download

Uma consulta SQL, requisitada a partir de uma função em PHP, pode ser salva num arquivo, comumente no formato de texto, e ser estruturada internamente de várias maneiras como, por exemplo, no formato CSV. Nesta sessão, serão descritos, sucintamente, a função desenvolvida para realizar essa solução e os testes para validá-la.

5.1.5.1 Geração do arquivo e *download*

Realizada a consulta, o recurso de download fica à disposição do usuário, porém, a tela de resultado somente será exibida caso respeitados os vários critérios para montar a consulta. No final da tabela, será encontrado o botão de "*Download .CSV"* mostrado na Figura 74.

O botão requisita a execução da função "*download($sql)"*, Figura 83, que está presente na *Model* "*Datawarehouse_model.php"*. Por parâmetro, a variável *$sql* recebe a consulta que foi gerada pelo mecanismo de consulta (sessão anterior, 5.1.4). A consulta fica armazenada numa variável da sessão para ser utilizada por este recurso quando solicitada e para não acionar novamente o conjunto de funções que gerou a *string* SQL. Dessa forma, haverá uma redução no custo de processamento.

Figura 83 – Função que gera o arquivo .CSV

```
public function download($sql){
    $dl = "\t"; //delimitador
    $nl = "\r\n"; //quebra de linha
    $encl = '"'; //cerca
    $this->load->dbutil();
    $this->load->helper('file');
    $this->load->helper('download');
    $filename = "report_".date('d-m-Y_h-i', time()).".csv"; //nome do arquivo
    $query = $this->db->query($sql);
    $data = $this->dbutil->csv_from_result($query, $dl, $nl, $encl);
    force_download($filename, $data);
}
```

Fonte: Da Rosa (2016).

A função é baseada no modelo disponível pela documentação[74] do *framework CodeIgniter*. Dentro da função há três itens que podem ser modificados sem prejudicar o funcionamento, observando-se que devem ser alterados seguindo os padrões de codificação de um arquivo texto. O delimitador, por exemplo, é o caractere que terá a

[74] Disponível em: <http://www.codeigniter.com/userguide3/database/utilities.html>. Acessado em: 23/11/2016.

função de separar as informações (colunas da consulta) e está definido para tabular (\t). Pode ser substituído por vírgula, ponto e vírgula ou outros caracteres que atendam às futuras necessidades e estejam dentro dos padrões utilizados pelos arquivos do formato CSV.

A quebra de linha está definida pelos caracteres '\r\n', que representam o padrão *Carriage Return* (CR, ou seja, retorno do carro) e *Line Feed* (LF, nova linha), utilizados por sistemas operacionais como Microsoft Windows e Apple OS X. O item cerca deve ser alterado em aspas (') ou aspas duplas ("). Esse caractere será posto no início de e no final de cada informação, com a função de enclausurar a informação, delimitando onde começa e termina a coluna.

A Figura 84 mostra uma parte de um arquivo gerado pela função. Nele pode-se observar que a primeira linha representa o cabeçalho, sendo a respectiva origem do dado ("nome da tabela_nome do campo"), enquanto as demais linhas correspondem aos dados da tela de resultado.

Figura 84 – Arquivo gerado pela função de download.

```
1  "family_name"   "gene_symbol"   "geneorganism_name" "geneorganism_tape5address" "
2  "Fusobacteria"  "rpsL"  "30S ribosomal protein S12" "80608" "80240" "2222"  "8023
3  "Fusobacteria"  "nusB"  "transcription antitermination protein NusB"    "126527"
4  "Fusobacteria"  "secY"  "preprotein translocase subunit SecY"   "132450"    "1311
5  "Fusobacteria"  "secY"  "preprotein translocase subunit SecY"   "132450"    "1311
6  "Fusobacteria"  "secY"  "preprotein translocase subunit SecY"   "132450"    "1311
7  "Fusobacteria"  "secY"  "preprotein translocase subunit SecY"   "132450"    "1311
8  "Fusobacteria"  "rplF"  "50S ribosomal protein L6"  "134598"    "134065"    "3462
9  "Fusobacteria"  "rplF"  "50S ribosomal protein L6"  "134598"    "134065"    "3462
10 "Fusobacteria"  "rplF"  "50S ribosomal protein L6"  "134598"    "134065"    "3462
11 "Fusobacteria"  "rplF"  "50S ribosomal protein L6"  "134598"    "134065"    "3462
12 "Fusobacteria"  "rplE"  "50S ribosomal protein L5"  "135909"    "135358"    "4500
13 "Fusobacteria"  "rplE"  "50S ribosomal protein L5"  "135909"    "135358"    "4500
14 "Fusobacteria"  "rplE"  "50S ribosomal protein L5"  "135909"    "135358"    "4500
15 "Fusobacteria"  "rplE"  "50S ribosomal protein L5"  "135909"    "135358"    "4500
16 "Fusobacteria"  "rplB"  "50S ribosomal protein L2"  "139772"    "138942"    "4167
17 "Fusobacteria"  "uppP"  "undecaprenyl pyrophosphate phosphatase"    "198054"    "
```

Fonte: Da Rosa (2016).

5.1.5.2 Testes

Os testes foram definidos da seguinte maneira: o arquivo texto deverá ter o mesmo número de linhas resultantes da pesquisa mais uma (o cabeçalho), sendo que

as linhas não devem estar repetidas e a mesma consulta deve ser executada no SGBD (nesse caso o pgAdmin3). No final de cada arquivo, durante a fase de testes, será inserida a consulta passada por parâmetro na variável *$sql*, assim viabilizando o teste no SGBD. O resultado do SGBD deve ser o mesmo que está exibido em tela e no arquivo. A Figura 85 demonstra uma consulta padrão que será utilizada pela maioria dos testes, exceto pela cláusula *WHERE,* que mudará nos testes seguintes.

Figura 85 – Consulta utilizada para os testes de download

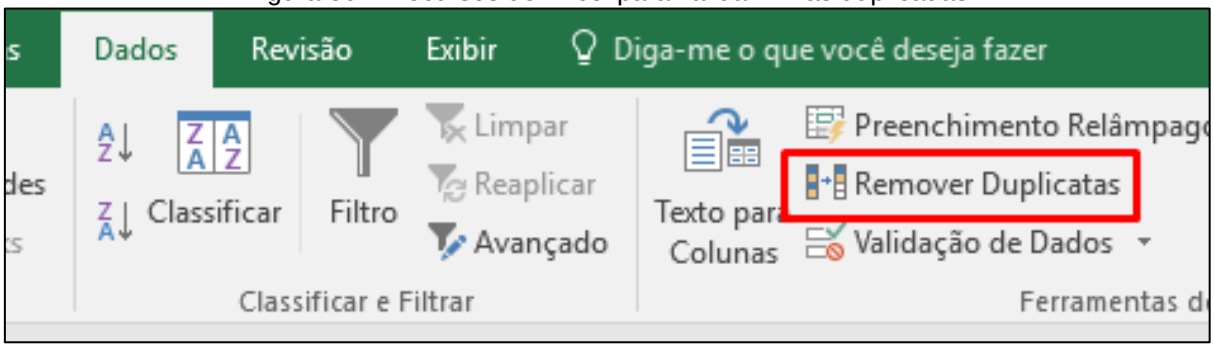
```
SELECT "f"."name" AS "family_name",
"g"."symbol" AS "gene_symbol",
"go"."name" AS "geneorganism_name",
"go"."tape5address" AS "geneorganism_tape5address",
"go"."tape3address" AS "geneorganism_tape3address",
"go"."gcpercentage" AS "geneorganism_gcpercentage",
"ir"."tape5address" AS "intergenicregion_tape5address",
"ir"."tape3address" AS "intergenicregion_tape3address",
"ir"."sequence" AS "intergenicregion_sequence",
"ir"."length" AS "intergenicregion_length",
"ir"."status" AS "intergenicregion_status",
"k"."name" AS "kingdom_name",
"o"."name" AS "organism_name",
"o"."dnamolecule" AS "organism_dnamolecule",
"r"."name" AS "role_name"
FROM "datawarehouse" AS "dw"
INNER JOIN "organism" AS "o" ON "dw"."organismid" = "o"."organismid"
INNER JOIN "family" AS "f" ON "dw"."familyid" = "f"."familyid"
INNER JOIN "gene" AS "g" ON "dw"."geneid" = "g"."geneid"
INNER JOIN "geneorganism" AS "go" ON "dw"."geneorganismid" = "go"."geneorganismid"
INNER JOIN "intergenicregion" AS "ir" ON "dw"."intergenicregionid" = "ir"."intergenicregionid"
INNER JOIN "kingdom" AS "k" ON "dw"."kingdomid" = "k"."kingdomid"
INNER JOIN "geneorganismrole" AS "gor" ON "dw"."geneorganismid" = "gor"."geneorganismid"
INNER JOIN "role" AS "r" ON "dw"."roleid" = "r"."roleid"
WHERE "o"."name" LIKE '%Fusobacterium%'
```

Fonte: Da Rosa (2016).

A verificação de linhas duplicadas será realizada através de outra ferramenta, o Microsoft Excel. O arquivo CSV será importado para uma planilha e executado o recurso para eliminar linhas duplicadas[75] (Figura 86). No final do processo ele exibe uma mensagem de alerta, informando quantas linhas foram removidas por estarem duplicadas e quantas restaram.

[75] Disponível em: <https://support.office.com/pt-br/article/Filtrar-valores-exclusivos-ou-remover-valores-duplicados-ccf664b0-81d6-449b-bbe1-8daaec1e83c2>. Acessado em: 23/11/2016.

Figura 86 – Recursos do Excel para validar linhas duplicadas

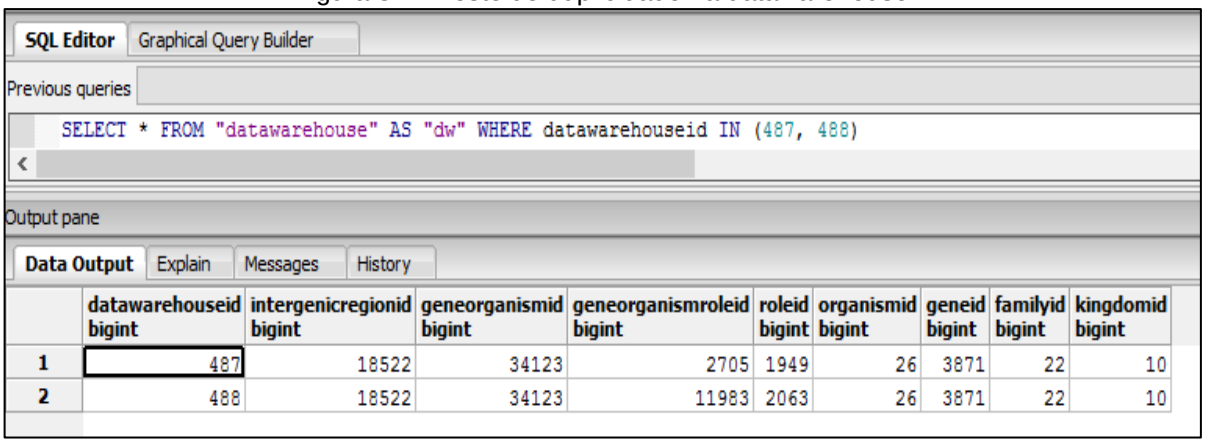


Fonte: Da Rosa (2016).

O primeiro teste é exatamente como na Figura 85, onde a clausula *WHERE* representa o que foi montado na tela de consulta (nome do organismo *LIKE* '%Fusobacterium%'). A contagem de linhas e os dados, visualmente, resultam em 122 linhas. Entretanto, ao executar a função de validação no Excel, foram apontadas apontou 42 linhas duplicadas. As mesmas estão presentes na tela, mas não são perceptíveis.

Por essa razão, a consulta no SGBD, foi alterada para mostrar todos os campos (*"SELEC * FOM..."*). O resultado foi importado para o Excel que não apresentou dados duplicados. As informações se distinguem das por outras colunas que não são visíveis na tela de resultado como, por exemplos, as *id* (números de identificação únicos). Durante a verificação da duplicidade de dados, constatou-se que a *datawarehouse* não está duplicando os dados (consulta executada na Figura 87).

Figura 87 – Teste de duplicidade na datawarehouse



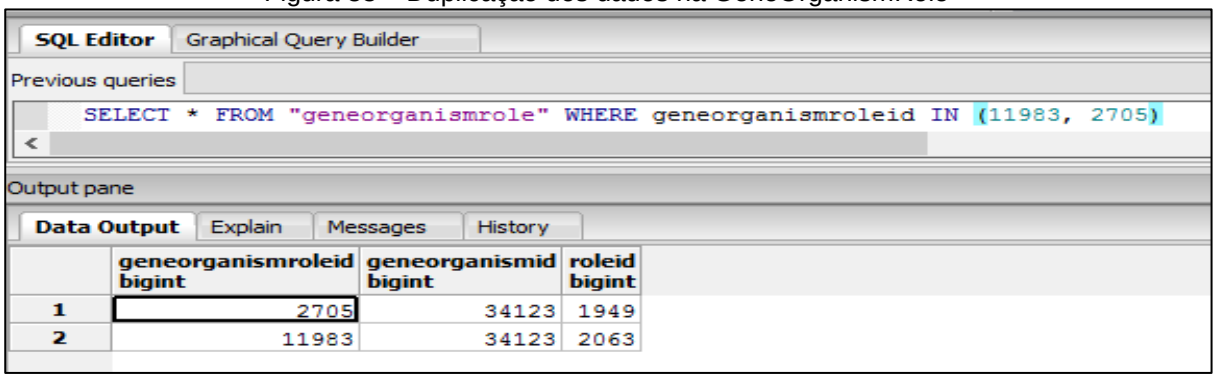
	datawarehouseid bigint	intergenicregionid bigint	geneorganismid bigint	geneorganismroleid bigint	roleid bigint	organismid bigint	geneid bigint	familyid bigint	kingdomid bigint
1	487	18522	34123	2705	1949	26	3871	22	10
2	488	18522	34123	11983	2063	26	3871	22	10

Fonte: Da Rosa (2016).

Ao continuar a verificação, localizou-se o ponto em que os dados estão duplicados. A consulta, realizada na Figura 88, mostra que o mesmo *GeneOrganismID*

está relacionado a mais de uma *GeneOrganismRoleID*. Segundo o usuário, o *GeneOrganismID* possui somente um *GeneOrganismRoleID*

Figura 88 – Duplicação dos dados na GeneOrganismRole



Fonte: Da Rosa (2016).

No teste de número 2, a clausula *WHERE* foi modificada para procurar famílias com o nome *"Aquificae"* ou com o nome *"Firmicutes"*. O resultado consiste em 24666 linhas na tela de resultado, no SGBD e no arquivo CSV gerado. O problema de duplicidade apresenta 6252 linhas duplicadas. Ao ser verificado, da mesma forma que o teste anterior, percebe-se que na verdade não estão duplos, pois se distinguem pelo *id* da mesma forma como ocorreu no teste 1.

O teste de número 3 pesquisa por todos os elementos que possuem a Fita inversa, ou seja, igual a "R". De um total de 60061 linhas, apresenta 18623 duplicadas, mesmo fato ocorrido nos testes 1 e 2. Os demais itens dos testes se mostram corretos.

Apesar da duplicação dos dados, a ferramenta está atendendo ao que foi proposto. O arquivo gerado corresponde ao requisitado pelo usuário na tela de consulta, ao visualizado na tela de pesquisa e também na tela do SGBD. O formato CSV está dentro dos padrões e é passível de ser importado para outras ferramentas, como o Microsoft Excel.

6 CONCLUSÃO

O portal IntergenicDB demonstrou-se um grande desafio por envolver a área da biologia molecular. Estando fora da minha zona de conhecimento, requisitou dedicação extra em compreender o assunto para melhor dialogar e compreender os usuários.

Durante a fase de análise do projeto, foram levantadas diversas problemáticas, tais como lentidão da base de dados, erros apresentados na tela de consulta, a inconsistência dos dados devido aos erros nos relacionamentos e outras. Os erros e apontamentos feitos pelo usuário geraram os requisitos para montar a proposta de solução deste trabalho.

No decorrer da execução, deparou-se com algumas limitações do servidor, tais como restrições de tempo para execução (tanto no banco de dados como no servidor web), limite de espaço em disco para execução (*cache*) e armazenamento, uma vez que utilizam da mesma área. Muitas dessas limitações não são da tecnologia ou da ferramenta, mas sim do provedor de serviço, decorrentes de políticas e restrições internas.

Buscando solucionar tais questões, foi realizada a evolução do portal para a versão 2.0, cuja equipe foi liderada pela formanda Camila Rachel Toni de Andrade. Essa evolução de software alterou a linguagem de programação (de ASP .NET C# para PHP), o banco de dados (de MySQL para PostgreSQL) e o *framework* (de *LP.Framework* para CodeIgniter). A arquitetura MVC e a POO foram mantidas, sendo que os requisitos da proposta deste trabalho ainda serão aplicados na nova versão.

Voltado a atender a proposta de solução, o *Data Warehouse* substitui a antiga *view* utilizada para consulta no banco de dados. Essa evolução trouxe melhorias de desempenho, confiabilidade e integridade aos dados.

Alguns apontamentos para melhorar, futuramente, o PromotoresDB ainda se fazem necessários como, por exemplo, definir métricas para o *Data Warehouse* e verificar a possibilidade de modificar alguns campos de nomes das tabelas *UNIQUE*, como, por exemplo, *Family.name, Kingdom.name, Gene.Symbol,* entre outros, recebendo restrições exclusivas (*Constraint Unique*). Tal pratica tornará a base mais íntegra e segura contra duplicações de dados e inserções de dados massiva. Dessa

forma, a responsabilidade de verificar a existência de um registro não é mais do software, mas sim do SGBD através de índices, o que é muito mais rápido e efetivo.

Outro tópico a ser verificado no PromotoresDB é a repetição de dados no resultado, causada pela inserção de dados na tabela *GeneOrganismRole*, mencionada na sessão 5.1.5.2.

O *script* gerado para a criação do *Data Warehouse* é necessário para a manutenção da tabela fato. Uma vez que novos dados são inseridos, cabe ao administrador do portal verificar a necessidade da execução desses comandos. Os novos dados não aparecerão na tela de resultado até a execução do *script*.

O novo mecanismo de busca trouxe melhorias de desempenho à consulta e às novas telas. O usuário dispõe de uma ferramenta mais versátil, dinâmica e segura para realizar as pesquisas. A melhor interação do usuário com a tela de pesquisa e seus recursos aumentou a facilidade de uso e compreensão da mesma. Se comparada à versão anterior, tem-se a possibilidade de realizar mais filtros, de acordo com a necessidade do usuário, tornando mais objetiva a consulta.

Já, a tela de resultado, está formatada para melhor atender à necessidade de leitura/análise dos dados em tela, permitindo que o usuário tenha diversos resultados, em diferentes abas do navegador *web*, sem perder as consultas realizadas anteriormente. A velocidade de leitura dos dados e impressão em tela se deve, além do *Data Warehouse*, ao uso de paginação. Filtrando o resultado em blocos, através dos recursos nativos do SQL e do *framework*, menores serão os recursos necessários para o processamento.

Como sugestão de melhoria, futura, pode-se atribuir recursos de filtragem dos dados da tela de resultado, assim como a possibilidade de ordenação dos resultados através das colunas visíveis.

Com a padronização do arquivo de download, demais necessidades do usuário podem ser sanadas através de softwares de terceiros que permitam manipular e formatar os dados de diferentes maneiras, as quais o portal ainda não viabiliza. Um recurso a ser mencionado para futuras melhorias é o desenvolvimento de um meio, ou ferramenta, que possibilite a extração de dados em massa da base PromotoresDB, função existente nos BDBM estudados nesse trabalho.

Após realizada a evolução do portal e as melhorias propostas, verificou-se que a tela não possui tamanho fixo. Atendendo os padrões atuais de desenvolvimento para *web*, a tela é responsiva, ou seja, adapta-se a diferentes tamanhos de monitores e

dispositivos como *smartphones*, *tablets* e TVs. Outra vantagem é que as melhorias utilizam menos da capacidade da banda (consumo de trafego da Internet do usuário) para a comunicação entre cliente-servidor.

6 REFERÊNCIAS

ALBERTS, Bruce et al. **Biologia Molecular da Célula.** 5. ed. São Paulo: Artmed, 2010. Disponível em: <http://integrada.minhabiblioteca.com.br/#/books/9788536321707/>. Acesso em: 15 ago. 2015.

ÁVILA E SILVA, Scheila de. **Redes neurais artificiais aplicadas no reconhecimento de regiões promotoras em bactérias Gram-negativas.** 2011. 108 f. Tese (Doutorado) - Curso de Pós-graduação em Biotecnologia, Centro de Ciências Agrárias e Biológicas, Universidade de Caxias do Sul, Caxias do Sul, 2011.

BIOTECNOLOGIA: Ciência e desenvolvimento. Rio de Janeiro: Kl3, 2002. Disponível em: <http://www.biotecnologia.com.br/revista/bio29/bioinf.pdf>. Acesso em: 26 nov 2015.

BROWN, Terence A.. **Genética:** Um enfoque molecular. 3. ed. Rio de Janeiro: Guanabara Koogan, 2009. Disponível em: <http://integrada.minhabiblioteca.com.br/#/books/978-85-277-2342-8/>. Acesso em: 15 ago. 2015.

CodeIgniter User Guide. Disponível em: <http://www.codeigniter.com/user_guide/>. Acessado em: 27/11/2017.

COX, Michael M.; DOUDNA, Jennifer A.; O'DONNEL, Michael. **Biologia molecular: princípios e técnicas.** Porto Alegre: Artmed, 2012. Disponível em: <http://integrada.minhabiblioteca.com.br/#/books/9788536327419/>. Acesso em: 10 ago. 2015.

DALZOCHIO, Jovani. **Implemenação de novas Funcionalidades para o portal lintergenicDB e povoamento do seu banco de dados**. 2014. 74 f. TCC (Graduação) - Curso de Bacharelado em Ciência da Computação, Universidade de Caxias do Sul, Caxias do Sul, 2014.

DAVANZO, Vanessa. **Desenvolvimento de Consultas para um Banco de Dados de Sequências Intergênicas.** 2010. 76 f. TCC (Graduação) - Curso de Bacharelado em Ciência da Computação, Universidade de Caxias do Sul, Caxias do Sul, 2010. Computação, Universidade de Caxias do Sul, Caxias do Sul, 2014.

DDBJ. **DNA Data Bank Japan.** Disponível em: <http://www.ddbj.nig.ac.jp/index-e.html>. Acessado em: 24/10/2015.

ELMASRI, Ramez; NAVATHE, Shamkant B.. **Sistemas de banco de dados**. 4. ed. São Paulo: Person, 2005

EMBL. **European Molecular Biology Laboratory.** Disponível em: <http://www.embl.de>. Acessado em: 12/10/2015.

FOWLER, Martin; RICE, David. **Padrões de arquitetura de aplicações corporativas**. Porto Alegre: Bookman, 2006.

GAMMA, Erich. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos.** São Paulo: Makron, 2000. Disponível em: <http://integrada.minhabiblioteca.com.br/#/books/9788577800469>. Acesso em : 31 out. 2015.

GOIS, Marcos de Meira.**Extração, mineração e carga dos dados - ETL.** 2004. 61 f. Monografia (Especialização) - Curso de Especialização: Novas Tecnologias em Sistemas Computacionais, Universidade de Caxias do Sul, Caxias do Sul, 2004.

GOWDAK, Demétrio Ossowski; MATTOS, Neide Simões de; PEZZI, Antônio Carlos. **Biologia:** Citologia, embriologia, histologia. São Paulo: Ftd, 2013. 3 v.

INTERGENICDB. **Banco de dados de regiões intergênicas de seres procariotos.** Disponível em: <http://intergenicdb.bioinfoucs.com/>. Acessado em: 25/10/2015.

KIMBALL, Ralph; ROSS, Margy. **The data warehouse toolkit:** Guia completo para modelagem dimensional. 2. ed. Rio de Janeiro: Campus, 2002.

LESK, Arthur M.. **Introdução a bioinformática.** 2. ed. Porto Alegre: Artmed, 2008.

LESSA, Felipe Almeida. **Identificação de RNAs não-codificadores por modelos de covariância com prioris Dirichlet adaptadas a grupos de ncRNAs com estruturas secundárias similares.** 2012. 97 f. Dissertação (Mestrado) - Curso de Mestrado em Informática, Universidade de Brasília, Brasília, 2012. Disponível em: <http://repositorio.unb.br/bitstream/10482/11840/1/2012_FelipeAlmeidaLessa.pdf>. Acesso em: 19 out. 2015.

MARÇULA, Marcelo; BENINI FILHO, Pio Armando. **Informática:** Conceitos e aplicações. 4. ed. São Paulo: Érica, 2013. Disponível em: <http://integrada.minhabiblioteca.com.br/#/books/9788536505343>. Acesso em: 19 out. 2015.

MOLIN, Aurione Francisco. **Prototipagem de um Banco de Dados de Promotores como base para um Portal de Serviços**. 2009. 74 f. TCC (Graduação) - Curso de Bacharelado em Ciência da Computação, Universidade de Caxias do Sul, Caxias do Sul, 2009.

MySQL Documentation. Disponível em: <http://dev.mysql.com/doc/>. Acessado em: 27/11/2017.

NCBI. **National Center for Biotechnology Information.** Disponível em: <http://www.ncbi.nlm.nih.gov>. Acessado em: 24/10/2015.

NOTARI, Daniel Luis et al (Org.). **IntergenicDB: a database for intergenic sequences. Bioinformation**. Caxias do Sul, p. 381-382. Maio 2014. Disponível em: <http://www.bioinformation.net/010/97320630010381.pdf>. Acesso em: 02 set. 2015.

_____. **Desenvolvimento de workflows científicos para a geração e análise de diferentes redes de interatomas**. 2012. xix, 161 f. Tese (Doutorado) - Universidade de Caxias do Sul, Programa de Pós-Graduação em Biotecnologia, 2012.

PICOLOTTO, Douglas. **Implementação de novas funcionalidades para o portal IntergenicDB.** 2012. 99 f. TCC (Graduação) - Curso de Curso de Bacharelado em Sistemas de Informação, Universidade de Caxias do Sul, Caxias do Sul, 2012.

PostgreSQL 8.3.23 Documentation. Disponível em: <https://www.postgresql.org/docs/8.3>. Acessado em: 27/11/2017.

reCAPTCHA. Protect your site from spam and abuse. Disponível em: <https://developers.google.com/recaptcha/>. Acessado em: 27/11/2017.

SADAVA, David et al. **Vida a ciência da biologia:** Volume 1: célula e hereditariedade**.** 8. ed. Porto Alegre: Artmed, 2009. 3 v.

SANCHES, Fabrício; ALTHMANN, Márcio Fábio. **Desenvolvimento web com ASP.NET VMC**. São Paulo: Casa do Código, 2013.

SANDERS, Mark F.; BOWMAN, John L.. **Análise genética:** Uma abordagem integrada. São Paulo: Person Education do Brasil, 2014. Disponível em: <http://ucs.bv3.digitalpages.com.br/users/publications/9788543005911>. Acesso em: 19 out. 2015.

SARTOR, João Carlos.**Utilização da mineração de dados na avaliação institucional da Universidade de Caxias do Sul - UCS.** 2004. 65 f. Monografia (Especialização) - Curso de Curso de Especialização em Novas Tecnologias Para O Desenvolvimento de Sistemas Computacionais, Universidade de Caxias do Sul, Caxias do Sul, 2004.

SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo: Pearson, 2011.
VERLI, Hugo (Org.). **Bioinformática: da biologia à flexibilidade molecula**r. São Paulo: Sbbq, 2014. Disponível em: <http://www.ufrgs.br/bioinfo/ebook/>. Acesso em: 29 ago. 2015.

ANDRADE, Camila Rachel Tonin de. Intergenicdb 2.0. 2016. 85 f. TCC (Graduação) - Curso de Bacharelado em Sistemas de Informação, Universidade de Caxias do Sul, Caxias do Sul, 2016.

VIEIRA, Rafael de Souza. **Implantação de business intelligence no sistema NL Gestão.** 2012. 100 f. TCC (Graduação) - Curso de Curso de Bacharelado em Sistemas de Informação, Universidade de Caxias do Sul, Caxias do Sul, 2012.

ZAHA, Arnaldo; FERREIRA, Henrique Bunselmeyer; PASSAGLIA, Luciane M. P.. **Biologia Molecular Básica.** 5. ed. Porto Alegre: Artmed, 2014. Disponível em: <http://integrada.minhabiblioteca.com.br/#/books/9788582710586>. Acesso em: 19 out. 2015

ANEXO A – *SCRIPT PARA CRIAR O PROMOTORESDB NO POSTGRESQL*

```
CREATE SCHEMA "public" AUTHORIZATION bacpp;
GRANT ALL ON SCHEMA public TO bacpp;
COMMENT ON SCHEMA public IS 'standard public schema';

set schema 'public';
CREATE TABLE public.family (
  familyid bigserial NOT NULL,
  name character varying(128),
  CONSTRAINT family_pkey PRIMARY KEY (familyid)
);
CREATE INDEX inxb_family_name ON public.family USING btree (name);
CREATE INDEX inxb_family_idname ON public.family USING btree (familyid, name);
CREATE INDEX inx_family_id ON public.family USING hash (familyid);
CREATE INDEX inxb_family_id ON public.family USING btree(familyid);

CREATE TABLE public.gene(
  geneid bigserial NOT NULL,
  symbol character varying(64),
  CONSTRAINT gene_pkey PRIMARY KEY (geneid)
);
CREATE INDEX inxb_gene_name ON public.gene USING btree (symbol);
CREATE INDEX inxb_gene_idname ON public.gene USING btree (geneid, symbol);
CREATE INDEX inx_gene_id ON public.gene USING hash (geneid);
CREATE INDEX inxb_gene_id ON public.gene USING btree(geneid);

CREATE TABLE public.role(
  roleid bigserial NOT NULL,
  name character varying(256) NOT NULL,
  CONSTRAINT role_pkey PRIMARY KEY (roleid)
);
CREATE INDEX inxb_role_name ON public.role USING btree (name);
CREATE INDEX inxb_role_idname ON public.role USING btree (roleid, name);
CREATE INDEX inx_role_id ON public.role USING hash (roleid);
CREATE INDEX inxb_role_id ON public.role USING btree(roleid);

CREATE TABLE kingdom(
  kingdomid bigserial NOT NULL,
  name character varying(64),
  CONSTRAINT kingdom_pkey PRIMARY KEY (kingdomid)
);
CREATE INDEX inxb_kingdom_name ON public.kingdom USING btree (name);
CREATE INDEX inxb_kingdom_idname ON public.kingdom USING btree (kingdomid, name);
CREATE INDEX inx_kingdom_id ON public.kingdom USING hash (kingdomid);
CREATE INDEX inxb_kingdom_id ON public.kingdom USING btree(kingdomid);


CREATE TABLE public.organism(
  organismid bigserial NOT NULL,
  kingdomid bigint NOT NULL,
  familyid bigint NOT NULL,
  name character varying(512) NOT NULL,
  dnamolecule character varying(512) NOT NULL,
  CONSTRAINT organism_pkey PRIMARY KEY (organismid),
  CONSTRAINT organism_familyid_fkey FOREIGN KEY (familyid)
      REFERENCES family (familyid) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
```

```
  CONSTRAINT organism_kingdomid_fkey FOREIGN KEY (kingdomid)
      REFERENCES kingdom (kingdomid) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE INDEX inxb_organism_name ON public.organism USING btree (name);
CREATE INDEX inxb_organism_dnamolecule
       ON public.organism USING btree (dnamolecule);
CREATE INDEX inxb_organism_idname
       ON public.organism USING btree (organismid, name);
CREATE INDEX inx_organism_id ON public.organism USING hash (organismid);
CREATE INDEX inxb_organism_id ON public.organism USING btree(organismid);
CREATE INDEX inxb_organism_fid
       ON public.organism USING btree (organismid, familyid);
CREATE INDEX inxb_organism_kid
       ON public.organism USING btree (organismid, kingdomid);
CREATE INDEX inxb_organism_fid_kid
        ON public.organism USING btree (organismid, familyid, kingdomid);

CREATE TABLE public.geneorganism(
  geneorganismid bigserial NOT NULL,
  geneid bigint NOT NULL,
  organismid bigint NOT NULL,
  name character varying(512),
  tape5address bigint NOT NULL,
  tape3address bigint NOT NULL,
  gcpercentage double precision,
  CONSTRAINT geneorganism_pkey PRIMARY KEY (geneorganismid),
  CONSTRAINT geneorganism_organismid_fkey FOREIGN KEY (organismid)
      REFERENCES organism (organismid) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT geneorganism_geneid_organismid_key
       UNIQUE (geneid, organismid));
CREATE INDEX inxb_geneorganism_name
       ON public.geneorganism USING btree (name);
CREATE INDEX inxb_geneorganism_idname
       ON public.geneorganism USING btree (geneorganismid, name);
CREATE INDEX inx_geneorganism_id
       ON public.geneorganism USING hash (geneorganismid);
CREATE INDEX inxb_geneorganism_id
       ON public.geneorganism USING btree(geneorganismid);

CREATE INDEX inxb_geneorganism_ids
       ON public.geneorganism USING btree (geneorganismid, geneid, organismid);
CREATE INDEX inxb_geneorganism_oid
       ON public.geneorganism USING btree (geneorganismid, organismid);
CREATE INDEX inxb_geneorganism_gid
       ON public.geneorganism USING btree (geneorganismid, geneid);
CREATE INDEX inxb_geneorganism_gid_oid
       ON public.geneorganism USING btree (geneid, organismid);

CREATE INDEX inxb_geneorganism_tp5
       ON public.geneorganism USING btree (tape5address);
CREATE INDEX inxb_geneorganism_tp3
       ON public.geneorganism USING btree (tape3address);
CREATE INDEX inxb_geneorganism_cg
       ON public.geneorganism USING btree (gcpercentage);
CREATE INDEX inx_geneorganism_tp5
       ON public.geneorganism USING hash (tape5address);
CREATE INDEX inx_geneorganism_tp3
```

```
      ON public.geneorganism USING hash (tape3address);
CREATE INDEX inx_geneorganism_cg ON public.geneorganism USING hash (gcpercentage);

CREATE TABLE public.geneorganismrole(
  geneorganismroleid bigserial NOT NULL,
  geneorganismid bigint NOT NULL,
  roleid bigint NOT NULL,
  CONSTRAINT geneorganismrole_pkey PRIMARY KEY (geneorganismroleid),
  CONSTRAINT geneorganismrole_geneorganismid_fkey FOREIGN KEY (geneorganismid)
      REFERENCES geneorganism (geneorganismid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT geneorganismrole_roleid_fkey FOREIGN KEY (roleid)
      REFERENCES role (roleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT geneorganismrole_geneid_organismid_roleid_key
      UNIQUE (geneorganismid, roleid)
);

CREATE INDEX inx_geneorganismrole_id
      ON public.geneorganismrole USING hash (geneorganismroleid);
CREATE INDEX inxb_geneorganismrole_id
      ON public.geneorganismrole USING btree(geneorganismroleid);
CREATE INDEX inxb_geneorganismrole_goid
      ON public.geneorganismrole USING btree(geneorganismroleid, geneorganismid);
CREATE INDEX inxb_geneorganismrole_rid
      ON public.geneorganismrole USING btree(geneorganismroleid, roleid);
CREATE INDEX inxb_geneorganismrole_goid_rid ON public.geneorganismrole
      USING btree(geneorganismroleid, geneorganismid, roleid);

CREATE TABLE public.intergenicregion(
  intergenicregionid bigserial NOT NULL,
  tape5address bigint NOT NULL,
  tape3address bigint NOT NULL,
  sequence text NOT NULL,
  length bigint NOT NULL,
  status character(1) NOT NULL,
  geneorganismid bigint NOT NULL,
  CONSTRAINT intergenicregion_pkey1 PRIMARY KEY (intergenicregionid),
  CONSTRAINT intergenicregion_geneorganismid_fkey FOREIGN KEY (geneorganismid)
      REFERENCES geneorganism (geneorganismid) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT intergenicregion_geneorganismid_key UNIQUE (geneorganismid)
);

CREATE INDEX inxb_intergenicregion_tp3
      ON public.intergenicregion USING btree (tape3address);
CREATE INDEX inxb_intergenicregion_tp5
      ON public.intergenicregion USING btree (tape5address);
CREATE INDEX inx_intergenicregion_tp3
      ON public.intergenicregion USING hash (tape3address);
CREATE INDEX inx_intergenicregion_tp5
      ON public.intergenicregion USING hash (tape5address);
CREATE INDEX inxb_intergenicregion_sequence
      ON public.intergenicregion USING btree (sequence);
CREATE INDEX inxb_intergenicregion_length
      ON public.intergenicregion USING btree (length);
CREATE INDEX inx_intergenicregion_length
      ON public.intergenicregion USING hash (length);
CREATE INDEX inxb_intergenicregion_goid
```

```
      ON public.intergenicregion USING btree (geneorganismid);
CREATE INDEX inxb_intergenicregion_ids
      ON public.intergenicregion USING btree (intergenicregionid, geneorganismid);
CREATE INDEX inx_intergenicregionid
      ON public.intergenicregion USING hash (intergenicregionid);
CREATE INDEX inxb_intergenicregion_id
      ON public.intergenicregion USING btree(intergenicregionid);
CREATE INDEX inxb_intergenicregion_status
      ON public.intergenicregion USING btree (status);
CREATE INDEX inx_intergenicregion_status
      ON public.intergenicregion USING hash (status);

CREATE TABLE public.admingroup(
  admingroupid bigserial NOT NULL,
  name character varying(100) NOT NULL,
  CONSTRAINT admingroup_pkey PRIMARY KEY (admingroupid)
);
CREATE INDEX inxb_admingroup_name ON public.admingroup USING btree (name);
CREATE INDEX inxb_admingroup_idname
      ON public.admingroup USING btree (admingroupid, name);
CREATE INDEX inx_admingroup_id ON public.admingroup USING hash (admingroupid);
CREATE INDEX inxb_admingroup_id ON public.admingroup USING btree (admingroupid);

CREATE TABLE public.country(
  countryid bigserial NOT NULL,
  name character varying(512) NOT NULL,
  CONSTRAINT country_pkey PRIMARY KEY (countryid)
);

CREATE INDEX inxb_country_name ON public.country USING btree (name);
CREATE INDEX inxb_country_idname ON public.country USING btree (countryid, name);
CREATE INDEX inx_country_id ON public.country USING hash (countryid);
CREATE INDEX inxb_country_id ON public.country USING btree (countryid);

CREATE TABLE public.users(
  userid bigserial NOT NULL,
  password character varying(32) NOT NULL,
  firstname character varying(100) NOT NULL,
  email character varying(255) NOT NULL,
  lastname character varying(100) NOT NULL,
  countryid bigint NOT NULL,
  active boolean NOT NULL DEFAULT true,
  upload boolean NOT NULL DEFAULT false,
  CONSTRAINT users_pkey PRIMARY KEY (userid),
  CONSTRAINT fk_user_country FOREIGN KEY (countryid)
      REFERENCES country (countryid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT useremail UNIQUE (email)
);
CREATE INDEX inxb_users_id ON public.users USING btree (userid);
CREATE INDEX inxb_users_email ON public.users USING btree (email);
CREATE INDEX inxb_users_countryid ON public.users USING btree (countryid);

CREATE TABLE public.connection(
  connectionid bigserial NOT NULL,
  accessdatetime timestamp(0) without time zone DEFAULT now(),
  ipaddress character varying(15) DEFAULT NULL::character varying,
  countryid bigint NOT NULL,
  CONSTRAINT connection_pkey PRIMARY KEY (connectionid),
```

```
  CONSTRAINT fk_connection_country FOREIGN KEY (countryid)
      REFERENCES country (countryid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);
CREATE INDEX inxb_connection_ids
      ON public.connection USING btree (connectionid, countryid);
CREATE INDEX inx_connection_id ON public.connection USING hash (connectionid);
CREATE INDEX inxb_connection_id ON public.connection USING btree (connectionid);
CREATE INDEX inxb_connection_countryid
      ON public.connection USING btree (countryid);


CREATE TABLE public.admin(
  adminid bigserial NOT NULL,
  userid bigint NOT NULL,
  phonenumber character varying(20) DEFAULT NULL::character varying,
  admingroupid integer NOT NULL,
  CONSTRAINT admin_pkey PRIMARY KEY (adminid),
  CONSTRAINT fk_admin_admingroup FOREIGN KEY (admingroupid)
      REFERENCES admingroup (admingroupid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_admin_user FOREIGN KEY (userid)
      REFERENCES users (userid) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT admin_userid_unique UNIQUE (userid)
);
CREATE INDEX inxb_admin_id ON public.admin USING btree (adminid);
CREATE INDEX inxb_admin_admingroupid ON public.admin USING btree (admingroupid);

CREATE TABLE public.admingrouppermission(
  admingrouppermissionid bigserial NOT NULL,
  admingroupid bigint NOT NULL,
  permissionkey character varying(50) NOT NULL,
  permissionlevel character varying(4) NOT NULL,
  CONSTRAINT admingrouppermission_pkey PRIMARY KEY (admingrouppermissionid),
  CONSTRAINT fk_admingrouppermission_admingroup FOREIGN KEY (admingroupid)
      REFERENCES admingroup (admingroupid) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE INDEX inxb_admingrouppermission_id
      ON public.admingrouppermission USING btree (admingrouppermissionid);
CREATE INDEX inxb_admingrouppermission_admingroupid
      ON public.admingrouppermission USING btree (admingroupid);

CREATE TABLE public.importqueue(
  importqueueid bigserial NOT NULL,
  creation timestamp without time zone NOT NULL,
  filename character varying(255) NOT NULL,
  filesize bigint NOT NULL,
  userid bigint NOT NULL,
  CONSTRAINT importqueue_pkey PRIMARY KEY (importqueueid),
  CONSTRAINT fk_importqueue_users FOREIGN KEY (userid)
      REFERENCES users (userid) MATCH SIMPLE
      ON UPDATE RESTRICT ON DELETE RESTRICT
);

CREATE INDEX inxb_importqueue_id
      ON public.importqueue USING btree (importqueueid);
CREATE INDEX inxb_importqueue_userid ON public.importqueue USING btree (userid);
```

```
CREATE INDEX inxb_importqueue_creation
       ON public.importqueue USING btree (creation);

CREATE TABLE public.importqueueentry(
  importqueueentryid bigserial NOT NULL,
  genename character varying(512),
  genesymbol character varying(512),
  genetape5address character varying(512),
  genetape3address character varying(512),
  intergenicregionlength bigint,
  genegcpercentage double precision,
  genemainrole character varying(512),
  organismkingdom character varying(512),
  organismfamily character varying(512),
  organismname character varying(512),
  organismdnamolecule character varying(512),
  intergenicregionsequence text,
  intergenicregiontape5address bigint,
  intergenicregiontape3address bigint,
  intergenicregionstatus character(1) NOT NULL,
  importqueueid bigint,
  CONSTRAINT importqueueentry_pkey PRIMARY KEY (importqueueentryid)
);
CREATE INDEX inxb_importqueueentry_id
       ON public.importqueueentry USING btree (importqueueentryid);
CREATE INDEX inxb_importqueueentry_importqueueid
       ON public.importqueueentry USING btree (importqueueid);
CREATE INDEX inxb_importqueueentry_ids
       ON public.importqueueentry USING btree (importqueueentryid, importqueueid);

CREATE TABLE public.intergenicslice (
  intergenicregionid bigint NOT NULL,
  "position" bigint NOT NULL,
  sequence character varying(300) NOT NULL,
  CONSTRAINT intergenicslice_pkey PRIMARY KEY ("position", intergenicregionid),
  CONSTRAINT fk_intergenicslice_intergenicregion FOREIGN KEY (intergenicregionid)
      REFERENCES intergenicregion (intergenicregionid) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE INDEX inxb_intergenicslice_intergenicregionid
       ON public.intergenicslice USING btree (intergenicregionid);
CREATE INDEX inxb_intergenicslice_pkey
       ON public.intergenicslice USING btree (intergenicregionid, "position");
CREATE INDEX inxb_intergenicslice_sequence
       ON public.intergenicslice USING btree (sequence);

CREATE TABLE public.login_token(
  id bigserial NOT NULL,
  token character varying(64) NOT NULL,
  user_id integer NOT NULL,
  expiration_date timestamp without time zone DEFAULT now(),
  login_from smallint NOT NULL,
  CONSTRAINT login_token_pkey PRIMARY KEY (id)
);

CREATE INDEX inxb_logintoken_id ON public.login_token USING btree (id);
CREATE INDEX inxb_logintoken_userid ON public.login_token USING btree (user_id);
```

```
CREATE TABLE public.publication(
  publicationid bigserial NOT NULL,
  description_portuguese text NOT NULL,
  description_english text,
  link character varying(300),
  CONSTRAINT publication_pkey PRIMARY KEY (publicationid)
);
CREATE INDEX inxb_publication_publicationid
       ON public.publication USING btree (publicationid);
CREATE INDEX inxb_publication_description_portuguese
       ON public.publication USING btree (description_portuguese);
CREATE INDEX inxb_publication_description_english
       ON public.publication USING btree (description_english);

CREATE TABLE public.predictormethod(
  predictormethodid bigserial NOT NULL,
  name character varying(100) NOT NULL,
  hitpercentage double precision NOT NULL,
  info character varying(500) NOT NULL,
  publicationid bigint,
  CONSTRAINT predictormethod_pkey PRIMARY KEY (predictormethodid),
  CONSTRAINT fk_predictormethod_publicationid FOREIGN KEY (publicationid)
      REFERENCES publication (publicationid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);
CREATE INDEX inxb_predictormethod_id
       ON public.predictormethod USING btree (predictormethodid);

CREATE TABLE public.predictormethod_intergenicslice(
  predictormethodid bigint NOT NULL,
  intergenicregionid bigint NOT NULL,
  "position" bigint NOT NULL,
  CONSTRAINT predictormethod_intergenicslice_pkey
       PRIMARY KEY (predictormethodid, intergenicregionid, "position"),
  CONSTRAINT fk_predictormethod_intergenicslice_intergenicslice
       FOREIGN KEY (intergenicregionid, "position")
      REFERENCES intergenicslice (intergenicregionid, "position")
       MATCH SIMPLE ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT fk_predictormethod_intergenicslice_predictormethodid
       FOREIGN KEY (predictormethodid)
      REFERENCES predictormethod (predictormethodid) MATCH SIMPLE
      ON UPDATE RESTRICT ON DELETE RESTRICT
);
CREATE INDEX inxb_predictormethod_intergenicslice_fkpredictormethod
       ON public.predictormethod_intergenicslice USING btree (predictormethodid);
CREATE INDEX inxb_predictormethod_intergenicslice_fkintergenicslice
       ON public.predictormethod_intergenicslice
       USING btree (intergenicregionid, "position");
CREATE INDEX inxb_predictormethod_intergenicslice_fks
       ON public.predictormethod_intergenicslice
       USING btree(predictormethodid, intergenicregionid, "position");


CREATE TABLE public.token(
  id bigserial NOT NULL,
  token character varying(64) NOT NULL,
  user_id integer NOT NULL,
  user_name character varying(255) NOT NULL,
```

```
  date_created timestamp without time zone DEFAULT now(),
  expiration_date timestamp without time zone NOT NULL,
  enabled smallint NOT NULL,
  used smallint NOT NULL,
  login_from integer NOT NULL,
  CONSTRAINT token_pkey PRIMARY KEY (id)
);
CREATE INDEX inxb_token_id ON public.token USING btree (id);


CREATE TABLE public.useractivationpending(
  userid bigint NOT NULL,
  activationkey character varying(50) NOT NULL,
  CONSTRAINT useractivationpending_pkey PRIMARY KEY (userid),
  CONSTRAINT fk_useractivationpending_users FOREIGN KEY (userid)
      REFERENCES users (userid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE,
  CONSTRAINT index_key UNIQUE (activationkey)
);

CREATE INDEX inxb_useractivationpending_userid
ON public.useractivationpending USING btree (userid);
```

ANEXO B – SCRIPT PARA CRIAR A *DATAWAREHOUSE*

```
set schema 'public';
IF (SELECT 1
    FROM   pg_class c
    JOIN   pg_namespace n
    ON    (c.relnamespace = n.oid)
    WHERE  n.nspname = 'public'
    AND    c.relname = 'inx_gorgeneorganismid')
BEGIN
   REINDEX INDEX public.inx_gorgeneorganismid;
END
ELSE
BEGIN
   CREATE INDEX inx_gorgeneorganismid ON public.geneorganismrole USING hash (geneorganismid);
END

IF (SELECT 1
    FROM   pg_class c
    JOIN   pg_namespace n
    ON    (c.relnamespace = n.oid)
    WHERE  n.nspname = 'public'
    AND    c.relname = 'inx_irgeneorganismid')
BEGIN
   REINDEX INDEX public.inx_irgeneorganismid;
END
ELSE
BEGIN
   CREATE INDEX inx_irgeneorganismid ON public.intergenicregion USING hash (geneorganismid);
END


VACUUM FULL ANALYZE public.intergenicregion;
ANALYZE VERBOSE public.intergenicregion;
VACUUM FULL ANALYZE public.geneorganismrole;
```

```
ANALYZE VERBOSE public.geneorganismrole;


DROP TABLE IF EXISTS public.intergenicregiontmp0;
SET enable_seqscan = ON;
CREATE TABLE public.intergenicregiontmp0 AS
SELECT ir.intergenicregionid,
       ir.geneorganismid,
       gor.geneorganismroleid,
       gor.roleid
  FROM public.intergenicregion AS ir
INNER JOIN public.geneorganismrole AS gor
USING (geneorganismid);

IF (SELECT 1
    FROM   pg_class c
    JOIN   pg_namespace n
    ON    (c.relnamespace = n.oid)
    WHERE  n.nspname = 'public'
    AND    c.relname = 'intergenicregiontmp0')
BEGIN
   CREATE INDEX index_tmp0_geneorganismid ON public.intergenicregiontmp0 USING hash (geneorganismid);

   IF (SELECT 1
       FROM   pg_class c
       JOIN   pg_namespace n
       ON    (c.relnamespace = n.oid)
       WHERE  n.nspname = 'public'
       AND    c.relname = 'inx_goid')
   BEGIN
     REINDEX INDEX public.inx_goid;
   END
   ELSE
   BEGIN
     CREATE INDEX inx_goid ON public.geneorganism USING hash (geneorganismid);
   END
```

```
VACUUM FULL ANALYZE public.intergenicregiontmp0;
ANALYZE VERBOSE public.intergenicregiontmp0;
VACUUM FULL ANALYZE public.geneorganism;
ANALYZE VERBOSE public.geneorganism;

DROP TABLE IF EXISTS public.intergenicregiontmp1;
SET enable_seqscan = OFF;
CREATE TABLE public.intergenicregiontmp1 AS
SELECT tmp.intergenicregionid,
       tmp.geneorganismid,
       tmp.geneorganismroleid,
       tmp.roleid,
       go.organismid,
       go.geneid
  FROM public.intergenicregiontmp0 AS tmp
INNER JOIN public.geneorganism AS go
USING (geneorganismid);

IF (SELECT 1
    FROM   pg_class c
    JOIN   pg_namespace n
    ON    (c.relnamespace = n.oid)
    WHERE  n.nspname = 'public'
    AND    c.relname = 'intergenicregiontmp1')
BEGIN
   CREATE INDEX index_tmp1_organismid ON public.intergenicregiontmp1 USING hash (organismid);

   IF (SELECT 1
       FROM   pg_class c
       JOIN   pg_namespace n
       ON    (c.relnamespace = n.oid)
       WHERE  n.nspname = 'public'
       AND    c.relname = 'inx_organismid')
   BEGIN
      REINDEX INDEX public.inx_organismid;
```

```
END
ELSE
BEGIN
   CREATE INDEX inx_organismid ON public.organism USING hash (organismid);
END

VACUUM FULL ANALYZE public.intergenicregiontmp0;
ANALYZE VERBOSE public.intergenicregiontmp0;
VACUUM FULL ANALYZE public.geneorganism;
ANALYZE VERBOSE public.geneorganism;

DROP TABLE IF EXISTS public.datawarehouse;
DROP TABLE IF EXISTS public.intergenicregiontmp2;
DROP SEQUENCE IF EXISTS public.datawarehouse_seq;
CREATE SEQUENCE public.datawarehouse_seq;
SET enable_seqscan = OFF;
CREATE TABLE public.intergenicregiontmp2 AS
SELECT NEXTVAL('datawarehouse_seq') AS datawarehouseid,
       tmp.intergenicregionid,
       tmp.geneorganismid,
       tmp.geneorganismroleid,
       tmp.roleid,
       tmp.organismid,
       tmp.geneid,
       o.familyid,
       o.kingdomid
  FROM public.intergenicregiontmp1 AS tmp
INNER JOIN public.organism AS o
USING (organismid);

IF (SELECT 1
    FROM   pg_class c
    JOIN   pg_namespace n
    ON    (c.relnamespace = n.oid)
    WHERE  n.nspname = 'public'
    AND    c.relname = 'intergenicregiontmp2')
```

```
BEGIN

  DROP TABLE IF EXISTS public.datawarehouse;
  ALTER TABLE public.intergenicregiontmp2 RENAME TO datawarehouse;

  ALTER TABLE public.datawarehouse ALTER datawarehouseid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER intergenicregionid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER geneorganismid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER geneorganismroleid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER roleid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER organismid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER geneid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER familyid SET NOT NULL;
  ALTER TABLE public.datawarehouse ALTER kingdomid SET NOT NULL;

  ALTER TABLE public.datawarehouse ADD PRIMARY KEY (datawarehouseid);
  ALTER TABLE public.datawarehouse ALTER COLUMN datawarehouseid SET DEFAULT nextval('datawarehouse_seq');
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(intergenicregionid)REFERENCES
     public.intergenicregion(intergenicregionid) MATCH SIMPLE ON UPDATE CASCADE ON DELETE CASCADE;
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(geneorganismid) REFERENCES public.geneorganism(geneorganismid)
     MATCH SIMPLE ON UPDATE CASCADE ON DELETE CASCADE;
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(geneorganismroleid) REFERENCES
     public.geneorganismrole(geneorganismroleid) MATCH SIMPLE ON UPDATE CASCADE ON DELETE CASCADE;
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(roleid) REFERENCES public.role(roleid) MATCH SIMPLE ON UPDATE CASCADE
     ON DELETE CASCADE;
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(organismid) REFERENCES public.organism(organismid) MATCH SIMPLE ON
     UPDATE CASCADE ON DELETE CASCADE;
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(geneid) REFERENCES public.gene(geneid) MATCH SIMPLE ON UPDATE CASCADE
     ON DELETE CASCADE;
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(familyid) REFERENCES public.family(familyid) MATCH SIMPLE ON UPDATE
     CASCADE ON DELETE CASCADE;
  ALTER TABLE public.datawarehouse ADD FOREIGN KEY(kingdomid) REFERENCES public.kingdom(kingdomid) MATCH SIMPLE ON UPDATE
     CASCADE ON DELETE CASCADE;


  CREATE UNIQUE INDEX index_dw_datawarehouseid ON public.datawarehouse USING btree(datawarehouseid);
```

```
  CREATE INDEX index_dw_intergenicregionid ON public.datawarehouse USING hash (intergenicregionid);
  CREATE INDEX index_dw_geneorganismid ON public.datawarehouse USING hash (geneorganismid);
  CREATE INDEX index_dw_geneorganismroleid ON public.datawarehouse USING hash (geneorganismroleid);
  CREATE INDEX index_dw_roleid ON public.datawarehouse USING hash (roleid);
  CREATE INDEX index_dw_organismid ON public.datawarehouse USING hash (organismid);
  CREATE INDEX index_dw_geneid ON public.datawarehouse USING hash (geneid);
  CREATE INDEX index_dw_familyid ON public.datawarehouse USING hash (familyid);
  CREATE INDEX index_dw_kingdomid ON public.datawarehouse USING hash (kingdomid);
  IF (SELECT 1 FROM (SELECT SUM (COL) AS SOMA FROM (SELECT 1 AS COL FROM   pg_class c JOIN   pg_namespace n

          ON    (c.relnamespace = n.oid)
          WHERE  n.nspname = 'public'
          AND    c.relname IN ('index_dw_datawarehouseid', 'index_dw_intergenicregionid',
            'index_dw_geneorganismid', 'index_dw_geneorganismroleid', 'index_dw_roleid',
            'index_dw_organismid', 'index_dw_geneid', 'index_dw_familyid',
            'index_dw_kingdomid')) AS existsindex) AS VERIFICA WHERE SOMA=9)

  BEGIN
     VACUUM FULL ANALYZE public.datawarehouse;
     ANALYZE VERBOSE public.datawarehouse;


     DROP TABLE IF EXISTS public.intergenicregiontmp1;
     DROP TABLE IF EXISTS public.intergenicregiontmp0;
     COMMIT;
     SET enable_seqscan = ON;
     PRINT '  * DATAWAREHOUSE CONCLUIDO COM SUCESSO';
  END
  ELSE
  BEGIN
     PRINT '  * ERRO AO CRIAR A DATAWAREHOUSE, EM datawarehouse';
  END
END
ELSE
BEGIN
  PRINT '  * ERRO AO CRIAR A DATAWAREHOUSE, EM intergenicregiontmp2';
```

```
     END
   END
   ELSE
   BEGIN
     PRINT '  * ERRO AO CRIAR A DATAWAREHOUSE, EM intergenicregiontmp1';
   END
END
ELSE
BEGIN
   PRINT '  * ERRO AO CRIAR A DATAWAREHOUSE, EM intergenicregiontmp0';
END
```

ANEXO C – SCRIPT PARA CRIAR A *VIEW*

```
set schema 'teste_med';
SET enable_seqscan = OFF;
CREATE OR REPLACE VIEW teste_med.simplesearch as (
   SELECT row_number() over (order by tmp.intergenicregionid),
          tmp.intergenicregionid AS "IR_ID",
          tmp.geneorganismid AS "IR_GO_ID",
          tmp.tape5address AS "IR_TP5",
          tmp.tape3address AS "IR_TP3",
          tmp.sequence AS "IR_SEQUENCE",
          tmp.length AS "IR_LENGTH",
          tmp.status AS "IR_STATUS",
          (SELECT r.roleid from teste_med.role r where r.roleid = tmp.roleid)
             AS "R_NAME",
          (SELECT g.symbol from teste_med.gene g where g.geneid = tmp.geneid)
             AS "G_NAME",
          tmp.name AS "GO_NAME",
          tmp.tp5 AS "GO_TP5",
          tmp.tp3 AS "GO_TP3",
          tmp.gcpercentage AS "GO_CG",
          (SELECT f.name from teste_med.family f where f.familyid = o.familyid)
             AS "F_NAME",
          (SELECT k.name from teste_med.kingdom k where k.kingdomid = o.kingdomid)
             AS "K_NAME",
          o.name AS "O_NAME",
          o.dnamolecule AS "O_DNAMOLECULE"
     FROM (SELECT tmp1.intergenicregionid,
             tmp1.geneorganismid,
             tmp1.tape5address,
             tmp1.tape3address,
             tmp1.sequence,
             tmp1.length,
             tmp1.status,
             tmp1.roleid,
             go.geneid,
             go.organismid,
             go.name,
             go.tape5address AS tp5,
             go.tape3address as tp3,
             go.gcpercentage
      FROM (SELECT ir.intergenicregionid,
              ir.geneorganismid,
              ir.tape5address,
              ir.tape3address,
              ir.sequence,
              ir.length,
              ir.status,
              gor.roleid
            FROM teste_med.intergenicregion AS ir,
                 teste_med.geneorganismrole AS gor
            WHERE ir.geneorganismid = gor.geneorganismid
            ) AS tmp1, teste_med.geneorganism AS go
      WHERE tmp1.geneorganismid = go.geneorganismid
      ) AS tmp, teste_med.organism AS o
   WHERE tmp.organismid = o.organismid
);
```

ANEXO D – SCRIPT DE TESTE DA SESSÃO 5.1.1.3

```
--Tamanho e número de linhas das tabelas
SELECT "Esquema", "Tabela", "Linhas", "Tamanho"
FROM (SELECT relid,
             schemaname AS "Esquema",
             relname AS "Tabela",
             pg_total_relation_size(relid) AS "Tamanho"
      FROM pg_catalog.pg_statio_user_tables
      WHERE relname IN ('family', 'gene', 'geneorganism', 'geneorganismrole',
'intergenicregion', 'kingdom', 'organism', 'role')
     ) t1
INNER JOIN
(SELECT relid,n_live_tup AS  "Linhas"
FROM pg_stat_all_tables) t2
ON t1.relid = t2.relid
ORDER BY 1,2

--1º teste
--view
set schema 'public';
SELECT * FROM public.simplesearch
SELECT count(*) FROM teste_grand.simplesearch

--dw
SELECT ir.tape5address AS ir_tape5address,
       ir.tape3address AS ir_tape3address,
       ir.sequence AS ir_sequence,
       ir.length AS ir_length,
       ir.status AS ir_status,
       go.name AS go_name,
       go.gcpercentage AS go_gcpercentage,
       go.tape3address AS gotape3address,
       go.tape5address AS go_tape5address,
       r.name AS role,
       f.name AS family,
       k.name AS kingdom,
       o.name AS o_name,
       o.dnamolecule AS o_dnamolecule,
       g.symbol
  FROM teste_med.datawarehouse AS dw
INNER JOIN teste_med.gene AS g ON dw.geneid = g.geneid
INNER JOIN teste_med.intergenicregion AS ir ON dw.geneorganismid =
ir.geneorganismid
INNER JOIN teste_med.organism AS o ON dw.organismid = o.organismid
INNER JOIN teste_med.geneorganism AS go ON dw.geneorganismid = go.geneorganismid
INNER JOIN teste_med.geneorganismrole AS gor ON dw.geneorganismid =
gor.geneorganismid
INNER JOIN teste_med.role AS r ON dw.roleid = r.roleid
INNER JOIN teste_med.family AS f ON dw.familyid = f.familyid
INNER JOIN teste_med.kingdom AS k ON dw.kingdomid = k.kingdomid
LIMIT 100;

--2º teste
--view
SELECT * FROM public.simplesearch
where "GO_NAME" in (select name from public.geneorganism)
AND "G_NAME" ILIKE '%b%'
```

```
ORDER BY "G_NAME"

--numero de linhas da view do schema teste_grand,
set schema 'teste_grand';
SELECT count("G_NAME") FROM teste_grand.simplesearch where "G_NAME" ILIKE '%b%'

--dw
SELECT ir.tape5address AS ir_tape5address,
       ir.tape3address AS ir_tape3address,
       ir.sequence AS ir_sequence,
       ir.length AS ir_length,
       ir.status AS ir_status,
       go.name AS go_name,
       go.gcpercentage AS go_gcpercentage,
       go.tape3address AS gotape3address,
       go.tape5address AS go_tape5address,
       r.name AS role,
       f.name AS family,
       k.name AS kingdom,
       o.name AS o_name,
       o.dnamolecule AS o_dnamolecule,
       g.symbol
  FROM teste_med.datawarehouse AS dw
INNER JOIN teste_med.gene AS g ON dw.geneid = g.geneid
INNER JOIN teste_med.intergenicregion AS ir ON dw.geneorganismid =
ir.geneorganismid
INNER JOIN teste_med.organism AS o ON dw.organismid = o.organismid
INNER JOIN teste_med.geneorganism AS go ON dw.geneorganismid = go.geneorganismid
INNER JOIN teste_med.geneorganismrole AS gor ON dw.geneorganismid =
gor.geneorganismid
INNER JOIN teste_med.role AS r ON dw.roleid = r.roleid
INNER JOIN teste_med.family AS f ON dw.familyid = f.familyid
INNER JOIN teste_med.kingdom AS k ON dw.kingdomid = k.kingdomid
WHERE g.symbol iLIKE '%b%'
ORDER BY g.symbol
LIMIT 100;

--3º teste
--view
select * from public.simplesearch
where "GO_TP5" BETWEEN ( cast( (select max("GO_TP5")from public.simplesearch) as
float)/3 )  AND ( cast( (select max("GO_TP5")from public.simplesearch) as
float)/3*2 )
and "GO_NAME" ilike '%a%'
or "GO_NAME" ilike '%1%'

--numero de linhas do grand,
select count(*) from teste_grand.simplesearch
where "GO_TP5" BETWEEN ( cast( (select max("GO_TP5")from teste_grand.simplesearch)
as float)/3 )  AND ( cast( (select max("GO_TP5")from teste_grand.simplesearch) as
float)/3*2 )
and "GO_NAME" ilike '%a%'
or "GO_NAME" ilike '%1%'

--dw
SELECT ir.tape5address AS ir_tape5address,
       ir.tape3address AS ir_tape3address,
       ir.sequence AS ir_sequence,
       ir.length AS ir_length,
```

```
       ir.status AS ir_status,
       go.name AS go_name,
       go.gcpercentage AS go_gcpercentage,
       go.tape3address AS gotape3address,
       go.tape5address AS go_tape5address,
       r.name AS role,
       f.name AS family,
       k.name AS kingdom,
       o.name AS o_name,
       o.dnamolecule AS o_dnamolecule,
       g.symbol
  FROM teste_med.datawarehouse AS dw
INNER JOIN teste_med.gene AS g ON dw.geneid = g.geneid
INNER JOIN teste_med.intergenicregion AS ir ON dw.geneorganismid =
ir.geneorganismid
INNER JOIN teste_med.organism AS o ON dw.organismid = o.organismid
INNER JOIN teste_med.geneorganism AS go ON dw.geneorganismid = go.geneorganismid
INNER JOIN teste_med.geneorganismrole AS gor ON dw.geneorganismid =
gor.geneorganismid
INNER JOIN teste_med.role AS r ON dw.roleid = r.roleid
INNER JOIN teste_med.family AS f ON dw.familyid = f.familyid
INNER JOIN teste_med.kingdom AS k ON dw.kingdomid = k.kingdomid
WHERE go.tape5address BETWEEN ( cast( (select max("GO_TP5")from
teste_med.simplesearch) as float)/3 )  AND ( cast( (select max("GO_TP5")from
teste_med.simplesearch) as float)/3*2 )
and go.name ilike '%a%'
or go.name ilike '%1%'
LIMIT 100;
```